

PAC Checker

Mathias Fleury and Daniela Kaufmann

December 14, 2021

Abstract

Generating and checking proof certificates is important to increase the trust in automated reasoning tools. In recent years formal verification using computer algebra became more important and is heavily used in automated circuit verification. An existing proof format which covers algebraic reasoning and allows efficient proof checking is the practical algebraic calculus. In this development, we present the verified checker Pastèque that is obtained by synthesis via the Refinement Framework.

This is the formalization going with our FMCAD'20 tool presentation [1].

Contents

1	Overview	2
2	Libraries	4
2.1	More Polynomials	4
2.2	More Ideals	18
3	Finite maps and multisets	19
3.1	Finite sets and multisets	19
3.2	Finite map and multisets	20
3.3	More Theorem about Loops	54
4	Specification of the PAC checker	57
4.1	Ideals	57
4.2	PAC Format	59
5	Hash-Map for finite mappings	67
5.1	Operations	68
5.2	Patterns	70
5.3	Mapping to Normal Hashmaps	70
6	Checker Algorithm	73
6.1	Specification	73
6.2	Algorithm	75
6.3	Full Checker	86
7	Polynomials of strings	88
7.1	Polynomials and Variables	88
7.2	Addition	90
7.3	Normalisation	92
7.4	Correctness	92

8	Terms	97
8.1	Ordering	97
8.2	Polynomials	98
8.3	Addition	100
8.4	Multiplication	107
8.5	Normalisation	113
8.6	Multiplication and normalisation	121
8.7	Correctness	122
9	Executable Checker	124
9.1	Definitions	124
9.2	Correctness	132
10	Various Refinement Relations	149
11	Hash Map as association list	156
11.1	Conversion from assoc to other map	158
12	Initial Normalisation of Polynomials	159
12.1	Sorting	159
12.2	Sorting applied to monomials	160
12.3	Lifting to polynomials	163
13	Code Synthesis of the Complete Checker	175
14	Correctness theorem	189

```

theory PAC-More-Poly
  imports HOL-Library.Poly-Mapping HOL-Algebra.Polynomials Polynomials.MPoly-Type-Class
  HOL-Algebra.Module HOL-Library.Countable-Set
begin

```

1 Overview

One solution to check circuit of multipliers is to use algebraic method, like producing proofs on polynomials. We are here interested in checking PAC proofs on the Boolean ring. The idea is the following: each variable represents an input or the output of a gate and we want to prove the bitwise multiplication of the input bits yields the output, namely the bitwise representation of the multiplication of the input (modulo 2^n where n is the number of bits of the circuit).

Algebraic proof systems typically reason over polynomials in a ring $\mathbb{K}[X]$, where the variables X represent Boolean values. The aim of an algebraic proof is to derive whether a polynomial f can be derived from a given set of polynomials $G = \{g_1, \dots, g_l\} \subseteq \mathbb{K}[X]$ together with the Boolean value constraints $B(X) = \{x_i^2 - x_i \mid x_i \in X\}$. In algebraic terms this means to show that the polynomial $f \in \langle G \cup B(X) \rangle$.

In our setting we set $\mathbb{K} = \mathbb{Z}$ and we treat the Boolean value constraints implicitly, i.e., we consider proofs in the ring $\mathbb{Z}[X]/\langle B(X) \rangle$ to admit shorter proofs

The checker takes as input 3 files:

1. an input file containing all polynomials that are initially present;
2. a target (or specification) polynomial ;

3. a “proof” file to check that contains the proof in PAC format that shows that the specification is in the ideal generated by the polynomials present initially.

Each step of the proof is either an addition of two polynomials previously derived, a multiplication from a previously derived polynomial and an arbitrary polynomial, and the deletion a derived polynomial.

One restriction on the proofs compared to generic PAC proofs is that $x^2 = x$ in the Boolean ring we are considering.

The checker can produce two outputs: valid (meaning that each derived polynomial in the proof has been correctly derived and the specification polynomial was also derived at some point [either in the proof or as input]) or invalid (without proven information what went wrong).

The development is organised as follows:

- `PAC_Specification.thy` (this file) contains the specification as described above on ideals without any peculiarities on the PAC proof format
- `PAC_Checker_Specification.thy` specialises to the PAC format and enters the non-determinism monad to prepare the subsequent refinements.
- `PAC_Checker.thy` contains the refined version where polynomials are represented as lists.
- `PAC_Checker_Synthesis.thy` contains the efficient implementation with imperative data structure like a hash set.
- `PAC_Checker_MLton.thy` contains the code generation and the command to compile the file with the ML compiler MLton.

Here is an example of a proof and an input file (taken from the appendix of our FMCAD paper [1], available at http://fmv.jku.at/pacheck_pasteque):

```

<res.input>          <res.proof>
1 x*y;              3 = fz, -z+1;
2 y*z-y-z+1;      4 * 3, y-1, -fz*y+fz-y*z+y+z-1;
                   5 + 2, 4, -fz*y+fz;
                   2 d;
                   4 d;
<res.target>       6 * 1, fz, fz*x*y;
-x*z+x;           1 d;
                   7 * 5, x, -fz*x*y+fz*x;
                   8 + 6, 7, fz*x;
                   9 * 3, x, -fz*x-x*z+x;
                   10 + 8, 9, -x*z+x;

```

Each line starts with a number that is used to index the (conclusion) polynomial. In the proof, there are four kind of steps:

1. `3 = fz, -z+1;` is an extension that introduces a new variable (in this case `fz`);

2. $4 * 3, y-1, -fz*y+fz-y*z+y+z-1$; is a multiplication of the existing polynomial with index 3 by the arbitrary polynomial $y-1$ and generates the new polynomial $-fz*y+fz-y*z+y+z-1$ with index 4;
3. $5 + 2, 4, -fz*y+fz$; is an addition of the existing polynomials with index 2 and 4 and generates the new polynomial $-fz*y+fz$ with index 5;
4. 1 d ; deletes the polynomial with index 1 and it cannot be reused in subsequent steps.

Remark that unlike DRAT checker, we do forward checking and check every derived polynomial. The target polynomial can also be part of the input file.

2 Libraries

2.1 More Polynomials

Here are more theorems on polynomials. Most of these facts are extremely trivial and should probably be generalised and moved to the Isabelle distribution.

lemma *Const₀-add*:

$\langle \text{Const}_0 (a + b) = \text{Const}_0 a + \text{Const}_0 b \rangle$
by transfer
(simp add: Const₀-def single-add)

lemma *Const-mult*:

$\langle \text{Const} (a * b) = \text{Const} a * \text{Const} b \rangle$
by transfer *(simp add: Const₀-def times-monomial-monomial)*

lemma *Const₀-mult*:

$\langle \text{Const}_0 (a * b) = \text{Const}_0 a * \text{Const}_0 b \rangle$
by transfer *(simp add: Const₀-def times-monomial-monomial)*

lemma *Const0[simp]*:

$\langle \text{Const} 0 = 0 \rangle$
by transfer *(simp add: Const₀-def)*

lemma *(in -) Const-uminus[simp]*:

$\langle \text{Const} (-n) = - \text{Const} n \rangle$
by transfer *(auto simp: Const₀-def monomial-uminus)*

lemma *[simp]: Const₀ 0 = 0*

$\langle \text{MPoly} 0 = 0 \rangle$
by *(auto simp: Const₀-def zero-mpoly-def)*

lemma *Const-add*:

$\langle \text{Const} (a + b) = \text{Const} a + \text{Const} b \rangle$
by transfer *(simp add: Const₀-def single-add)*

instance *mpoly* :: *(comm-semiring-1) comm-semiring-1*

by standard

lemma *degree-uminus[simp]*:

$\langle \text{degree} (-A) x' = \text{degree} A x' \rangle$
by *(auto simp: degree-def uminus-mpoly.rep-eq)*

lemma *degree-sum-notin*:

$\langle x' \notin \text{vars } B \implies \text{degree } (A + B) x' = \text{degree } A x' \rangle$

apply (*auto simp: degree-def*)

apply (*rule arg-cong[of - - Max]*)

apply *standard+*

apply (*auto simp: plus-mpoly.rep-eq UN-I UnE image-iff in-keys-iff subsetD vars-def lookup-add*
dest: keys-add intro: in-keys-plusI1 cong: ball-cong-simp)

done

lemma *degree-notin-vars*:

$\langle x \notin (\text{vars } B) \implies \text{degree } (B :: 'a :: \{\text{monoid-add}\} \text{mpoly}) x = 0 \rangle$

using *degree-sum-notin[of x B 0]*

by *auto*

lemma *not-in-vars-coeff0*:

$\langle x \notin \text{vars } p \implies \text{MPoly-Type.coeff } p (\text{monomial } (\text{Suc } 0) x) = 0 \rangle$

by (*subst not-not[symmetric], subst coeff-keys[symmetric]*)

(*auto simp: vars-def*)

lemma *keys-add'*:

$p \in \text{keys } (f + g) \implies p \in \text{keys } f \cup \text{keys } g$

by *transfer auto*

lemma *keys-mapping-sum-add*:

$\langle \text{finite } A \implies \text{keys } (\text{mapping-of } (\sum v \in A. f v)) \subseteq \bigcup (\text{keys } ' \text{ mapping-of } ' f ' \text{ UNIV}) \rangle$

by (*induction A rule: finite-induct*)

(*auto simp add: zero-mpoly.rep-eq plus-mpoly.rep-eq*
keys-plus-ninv-comm-monoid-add dest: keys-add')

lemma *vars-sum-vars-union*:

fixes $f :: \langle \text{int mpoly} \Rightarrow \text{int mpoly} \rangle$

assumes $\langle \text{finite } \{v. f v \neq 0\} \rangle$

shows $\langle \text{vars } (\sum v \mid f v \neq 0. f v * v) \subseteq \bigcup (\text{vars } ' \{v. f v \neq 0\}) \cup \bigcup (\text{vars } ' f ' \{v. f v \neq 0\}) \rangle$
(*is* $\langle ?A \subseteq ?B \rangle$)

proof

fix p

assume $\langle p \in \text{vars } (\sum v \mid f v \neq 0. f v * v) \rangle$

then obtain x **where** $\langle x \in \text{keys } (\text{mapping-of } (\sum v \mid f v \neq 0. f v * v)) \rangle$ **and**

$p: \langle p \in \text{keys } x \rangle$

by (*auto simp: vars-def times-mpoly.rep-eq simp del: keys-mult*)

then have $\langle x \in (\bigcup x. \text{keys } (\text{mapping-of } (f x) * \text{mapping-of } x)) \rangle$

using *keys-mapping-sum-add*[of $\langle \{v. f v \neq 0\} \rangle \langle \lambda x. f x * x \rangle$] *assms*

by (*auto simp: vars-def times-mpoly.rep-eq*)

then have $\langle x \in (\bigcup x. \{a+b \mid a b. a \in \text{keys } (\text{mapping-of } (f x)) \wedge b \in \text{keys } (\text{mapping-of } x)\}) \rangle$

using *Union-mono[OF] keys-mult* **by** *fast*

then show $\langle p \in ?B \rangle$

using p **by** (*force simp: vars-def zero-mpoly.rep-eq dest!: keys-add'*)

qed

lemma *vars-in-right-only*:

$x \in \text{vars } q \implies x \notin \text{vars } p \implies x \in \text{vars } (p+q)$

unfolding *vars-def keys-def plus-mpoly.rep-eq lookup-plus-fun*

apply *clarify*

```

subgoal for  $xa$ 
  by (auto simp: vars-def keys-def plus-mpoly.rep-eq
    lookup-plus-fun intro!: exI[of - xa] dest!: spec[of - xa])
done

```

```

lemma [simp]:
   $\langle vars\ 0 = \{\} \rangle$ 
  by (simp add: vars-def zero-mpoly.rep-eq)

```

```

lemma vars-Un-nointer:
   $\langle keys\ (mapping-of\ p) \cap keys\ (mapping-of\ q) = \{\} \implies vars\ (p + q) = vars\ p \cup vars\ q \rangle$ 
  by (auto simp: vars-def plus-mpoly.rep-eq simp flip: More-MPoly-Type.keys-add dest!: keys-add')

```

```

lemmas [simp] = zero-mpoly.rep-eq

```

```

lemma polynomial-sum-monoms:
  fixes  $p :: \langle 'a :: \{comm-monoid-add, cancel-comm-monoid-add\} mpoly \rangle$ 
  shows
     $\langle p = (\sum x \in keys\ (mapping-of\ p). MPoly-Type.monom\ x\ (MPoly-Type.coeff\ p\ x)) \rangle$ 
     $\langle keys\ (mapping-of\ p) \subseteq I \implies finite\ I \implies p = (\sum x \in I. MPoly-Type.monom\ x\ (MPoly-Type.coeff\ p\ x)) \rangle$ 

```

proof –

```

define  $J$  where  $\langle J \equiv keys\ (mapping-of\ p) \rangle$ 

```

```

define  $a$  where  $\langle a\ x \equiv coeff\ p\ x \rangle$  for  $x$ 

```

```

have  $\langle finite\ (keys\ (mapping-of\ p)) \rangle$ 

```

```

  by auto

```

```

have  $\langle p = (\sum x \in I. MPoly-Type.monom\ x\ (MPoly-Type.coeff\ p\ x)) \rangle$ 

```

```

  if  $\langle finite\ I \rangle$  and  $\langle keys\ (mapping-of\ p) \subseteq I \rangle$ 

```

```

  for  $I$ 

```

```

  using that

```

```

  unfolding a-def

```

```

proof (induction I arbitrary: p rule: finite-induct)

```

```

  case empty

```

```

    then have  $\langle p = 0 \rangle$ 

```

```

      using empty coeff-all-0 coeff-keys by blast

```

```

    then show ?case using empty by (auto simp: zero-mpoly.rep-eq)

```

```

  next

```

```

    case (insert x F) note  $fin = this(1)$  and  $xF = this(2)$  and  $IH = this(3)$  and

```

```

       $incl = this(4)$ 

```

```

    let  $?p = \langle p - MPoly-Type.monom\ x\ (MPoly-Type.coeff\ p\ x) \rangle$ 

```

```

    have  $H: \langle \bigwedge xa. x \notin F \implies xa \in F \implies$ 

```

```

       $MPoly-Type.monom\ xa\ (MPoly-Type.coeff\ (p - MPoly-Type.monom\ x\ (MPoly-Type.coeff\ p\ x))$ 

```

```

 $xa) =$ 

```

```

       $MPoly-Type.monom\ xa\ (MPoly-Type.coeff\ p\ xa) \rangle$ 

```

```

      by (metis (mono-tags, opaque-lifting) add-diff-cancel-right' remove-term-coeff
        remove-term-sum when-def)

```

```

    have  $\langle ?p = (\sum xa \in F. MPoly-Type.monom\ xa\ (MPoly-Type.coeff\ ?p\ xa)) \rangle$ 

```

```

      apply (rule IH)

```

```

      using incl apply –

```

```

      by standard (smt Diff-iff Diff-insert-absorb add-diff-cancel-right'

```

```

        remove-term-keys remove-term-sum subsetD xF)

```

```

    also have  $\langle \dots = (\sum xa \in F. MPoly-Type.monom\ xa\ (MPoly-Type.coeff\ p\ xa)) \rangle$ 

```

```

      by (use xF in <auto intro!: sum.cong simp: H>)

```

```

    finally show ?case

```

```

apply (subst (asm) remove-term-sum[of x p, symmetric])
apply (subst remove-term-sum[of x p, symmetric])
using xF fin by (auto simp: ac-simps)
qed
from this[of I] this[of J] show
  ⟨p = (∑ x∈keys (mapping-of p). MPoly-Type.monom x (MPoly-Type.coeff p x))⟩
  ⟨keys (mapping-of p) ⊆ I ⟹ finite I ⟹ p = (∑ x∈I. MPoly-Type.monom x (MPoly-Type.coeff p
x))⟩
by (auto simp: J-def)
qed

```

lemma vars-mult-monom:

```

fixes p :: ⟨int mpoly⟩
shows ⟨vars (p * (monom (monomial (Suc 0) x') 1)) = (if p = 0 then {} else insert x' (vars p))⟩
proof -
  let ?v = ⟨monom (monomial (Suc 0) x') 1⟩
  have
    p: ⟨p = (∑ x∈keys (mapping-of p). MPoly-Type.monom x (MPoly-Type.coeff p x))⟩ (is ⟨- = (∑ x ∈
?I. ?f x)⟩)
    using polynomial-sum-monoms(1)[of p] .
  have pv: ⟨p * ?v = (∑ x ∈ ?I. ?f x * ?v)⟩
    by (subst p) (auto simp: field-simps sum-distrib-left)
  define I where ⟨I ≡ ?I⟩
  have in-keysD: ⟨x ∈ keys (mapping-of (∑ x∈I. MPoly-Type.monom x (h x))) ⟹ x ∈ I⟩
  if ⟨finite I⟩ for I and h :: ⟨- ⇒ int⟩ and x
  using that by (induction rule: finite-induct)
  (force simp: monom.rep-eq empty-iff insert-iff keys-single coeff-monom
simp: coeff-keys simp flip: coeff-add
simp del: coeff-add)+
  have in-keys: ⟨keys (mapping-of (∑ x∈I. MPoly-Type.monom x (h x))) = (∪ x ∈ I. (if h x = 0 then
{} else {x}))⟩
  if ⟨finite I⟩ for I and h :: ⟨- ⇒ int⟩ and x
  supply in-keysD[dest]
  using that by (induction rule: finite-induct)
  (auto simp: plus-mpoly.rep-eq MPoly-Type-Class.keys-plus-eqI)

  have H[simp]: ⟨vars ((∑ x∈I. MPoly-Type.monom x (h x))) = (∪ x∈I. (if h x = 0 then {} else keys
x))⟩
  if ⟨finite I⟩ for I and h :: ⟨- ⇒ int⟩
  using that by (auto simp: vars-def in-keys)

  have sums: ⟨(∑ x∈I.
    MPoly-Type.monom (x + a') (c x)) =
    (∑ x ∈ (λx. x + a') ' I.
    MPoly-Type.monom x (c (x - a'))))⟩
  if ⟨finite I⟩ for I a' c q
  using that apply (induction rule: finite-induct)
  subgoal by auto
  subgoal
    unfolding image-insert by (subst sum.insert) auto
  done
  have non-zero-keysEx: ⟨p ≠ 0 ⟹ ∃ a. a ∈ keys (mapping-of p)⟩ for p :: ⟨int mpoly⟩
  using mapping-of-inject by (fastforce simp add: ex-in-conv)

```

```

have ⟨finite I⟩ ⟨keys (mapping-of p) ⊆ I⟩
  unfolding I-def by auto
then show
  ⟨vars (p * (monom (monomial (Suc 0) x') 1)) = (if p = 0 then {} else insert x' (vars p))⟩
  apply (subst pv, subst I-def[symmetric], subst mult-monom)
  apply (auto simp: mult-monom sums I-def)
  using Poly-Mapping.keys-add vars-def apply fastforce
  apply (auto dest!: non-zero-keysEx)
  apply (rule-tac x= ⟨a + monomial (Suc 0) x'⟩ in beXI)
  apply (auto simp: coeff-keys)
  apply (simp add: in-keys-iff lookup-add)
  apply (auto simp: vars-def)
  apply (rule-tac x= ⟨xa + monomial (Suc 0) x'⟩ in beXI)
  apply (auto simp: coeff-keys)
  apply (simp add: in-keys-iff lookup-add)
done
qed

```

```

term ⟨(x', u, lookup u x', A)⟩
lemma in-mapping-mult-single:
  ⟨x ∈ (λx. lookup x x') 'keys (A * (Var0 x' :: (nat ⇒0 nat) ⇒0 'b :: {monoid-mult, zero-neq-one, semiring-0}))⟩
  ←→
  x > 0 ∧ x - 1 ∈ (λx. lookup x x') 'keys (A)⟩
  apply (standard+; clarify)
  subgoal
    apply (auto elim!: in-keys-timesE simp: lookup-add)
    apply (auto simp: keys-def lookup-times-monomial-right Var0-def lookup-single image-iff)
  done
  subgoal
    apply (auto elim!: in-keys-timesE simp: lookup-add)
    apply (auto simp: keys-def lookup-times-monomial-right Var0-def lookup-single image-iff)
  done
  subgoal for xa
    apply (auto elim!: in-keys-timesE simp: lookup-add)
    apply (auto simp: keys-def lookup-times-monomial-right Var0-def lookup-single image-iff lookup-add
      intro!: exI[of - ⟨xa + Poly-Mapping.single x' 1⟩])
  done
done

```

```

lemma Max-Suc-Suc-Max:
  ⟨finite A ⇒ A ≠ {} ⇒ Max (insert 0 (Suc 'A)) =
  Suc (Max (insert 0 A))⟩
  by (induction rule: finite-induct)
  (auto simp: hom-Max-commute)

```

```

lemma [simp]:
  ⟨keys (Var0 x' :: ('a ⇒0 nat) ⇒0 'b :: {zero-neq-one}) = {Poly-Mapping.single x' 1}⟩
  by (auto simp: Var0-def)

```

```

lemma degree-mult-Var:
  ⟨degree (A * Var x') x' = (if A = 0 then 0 else Suc (degree A x'))⟩ for A :: ⟨int mpoly⟩
proof -
  have [simp]: ⟨A ≠ 0 ⇒

```


$Max (insert\ 0\ ((\lambda x. Suc\ (lookup\ x\ x'))\ 'keys\ (mapping-of\ A))) =$
 $Max (insert\ (Suc\ 0)\ ((\lambda x. Suc\ (lookup\ x\ x'))\ 'keys\ (mapping-of\ A)))\rangle$
unfolding *image-image*[of *Suc* $\langle \lambda x. lookup\ x\ x' \rangle$, *symmetric*] *image-insert*[*symmetric*]
by (*subst* *Max-Suc-Suc-Max*, *use mapping-of-inject in fastforce*, *use mapping-of-inject in fastforce*) +
(simp add: Max.hom-commute)
have $\langle A \neq 0 \implies$
 $Max (insert\ 0$
 $((\lambda x. lookup\ x\ x')\ 'keys\ (mapping-of\ A * mapping-of\ (Var\ x')))) =$
 $Suc (Max (insert\ 0\ ((\lambda m. lookup\ m\ x')\ 'keys\ (mapping-of\ A))))\rangle$
by (*subst arg-cong*[of - $\langle insert\ 0$
 $(Suc\ '((\lambda x. lookup\ x\ x')\ 'keys\ (mapping-of\ A)))\rangle$ *Max*]
(auto simp: image-image Var.rep-eq lookup-plus-fun in-mapping-mult-single
hom-Max-commute Max-Suc-Suc-Max
elim!: in-keys-timesE split: if-splits)
then show *?thesis*
by (*auto simp: degree-def times-mpoly.rep-eq*
intro!: arg-cong[of - $\langle insert\ 0$
 $(Suc\ '((\lambda x. lookup\ x\ x')\ 'keys\ (mapping-of\ A)))\rangle$ *Max*]
qed

lemma *degree-mult-Var'*:
 $\langle degree\ (Var\ x' * A)\ x' = (if\ A = 0\ then\ 0\ else\ Suc\ (degree\ A\ x')) \rangle$ **for** $A :: \langle int\ mpoly \rangle$
by (*simp add: degree-mult-Var semiring-normalization-rules*(γ))

lemma *degree-times-le*:
 $\langle degree\ (A * B)\ x \leq degree\ A\ x + degree\ B\ x \rangle$
by (*auto simp: degree-def times-mpoly.rep-eq*
max-def lookup-add add-mono
dest!: set-rev-mp[OF - Poly-Mapping.keys-add]
elim!: in-keys-timesE)

lemma *monomial-inj*:
 $monomial\ c\ s = monomial\ (d::'b::zero-neq-one)\ t \iff (c = 0 \wedge d = 0) \vee (c = d \wedge s = t)$
by (*fastforce simp add: monomial-inj Poly-Mapping.single-def*
poly-mapping.Abs-poly-mapping-inject when-def fun-eq-iff
cong: if-cong
split: if-splits)

lemma *MPoly-monomial-power'*:
 $\langle MPoly\ (monomial\ 1\ x')\ ^{(n+1)} = MPoly\ (monomial\ (1)\ (((\lambda x. x + x')\ \overset{\sim}{\sim} n)\ x')) \rangle$
by (*induction n*)
(auto simp: times-mpoly.abs-eq mult-single ac-simps)

lemma *MPoly-monomial-power*:
 $\langle n > 0 \implies MPoly\ (monomial\ 1\ x')\ ^{(n)} = MPoly\ (monomial\ (1)\ (((\lambda x. x + x')\ \overset{\sim}{\sim} (n - 1))\ x')) \rangle$
using *MPoly-monomial-power'*[of - $\langle n-1 \rangle$]
by *auto*

lemma *vars-uminus*[*simp*]:
 $\langle vars\ (-p) = vars\ p \rangle$
by (*auto simp: vars-def uminus-mpoly.rep-eq*)

lemma *coeff-uminus*[*simp*]:

$\langle \text{MPoly-Type.coeff } (-p) \ x = -\text{MPoly-Type.coeff } p \ x \rangle$
by (auto simp: coeff-def uminus-mpoly.rep-eq)

definition decrease-key:: $'a \Rightarrow ('a \Rightarrow_0 'b :: \{\text{monoid-add, minus, one}\}) \Rightarrow ('a \Rightarrow_0 'b)$ **where**
 decrease-key $k0 \ f = \text{Abs-poly-mapping } (\lambda k. \text{if } k = k0 \wedge \text{lookup } f \ k \neq 0 \text{ then } \text{lookup } f \ k - 1 \text{ else } \text{lookup } f \ k)$

lemma remove-key-lookup:

lookup (decrease-key $k0 \ f$) $k = (\text{if } k = k0 \wedge \text{lookup } f \ k \neq 0 \text{ then } \text{lookup } f \ k - 1 \text{ else } \text{lookup } f \ k)$

unfolding decrease-key-def **using** finite-subset **apply** (simp add:)

apply (subst lookup-Abs-poly-mapping)

apply (auto intro: finite-subset[of - $\langle \{x. \text{lookup } f \ x \neq 0\} \rangle$])

apply (subst lookup-Abs-poly-mapping)

apply (auto intro: finite-subset[of - $\langle \{x. \text{lookup } f \ x \neq 0\} \rangle$])

done

lemma polynomial-split-on-var:

fixes $p :: 'a :: \{\text{comm-monoid-add, cancel-comm-monoid-add, semiring-0, comm-semiring-1}\} \ \text{mpoly}$

obtains $q \ r$ **where**

$\langle p = \text{monom } (\text{monomial } (\text{Suc } 0) \ x') \ 1 * q + r \rangle$ **and**

$\langle x' \notin \text{vars } r \rangle$

proof –

have [simp]: $\langle \{x \in \text{keys } (\text{mapping-of } p). \ x' \in \text{keys } x\} \cup \{x \in \text{keys } (\text{mapping-of } p). \ x' \notin \text{keys } x\} = \text{keys } (\text{mapping-of } p) \rangle$

by auto

have

$\langle p = (\sum_{x \in \text{keys } (\text{mapping-of } p)}. \ \text{MPoly-Type.monom } x \ (\text{MPoly-Type.coeff } p \ x)) \rangle$ (**is** $\langle - = (\sum_{x \in ?I}. \ ?f \ x) \rangle$)

using polynomial-sum-monoms(1)[of p].

also have $\langle \dots = (\sum_{x \in \{x \in ?I. \ x' \in \text{keys } x\}. \ ?f \ x} + (\sum_{x \in \{x \in ?I. \ x' \notin \text{keys } x\}. \ ?f \ x}) \rangle$ (**is** $\langle - = ?pX + ?qX \rangle$)

by (subst comm-monoid-add-class.sum.union-disjoint[symmetric]) auto

finally have 1: $\langle p = ?pX + ?qX \rangle$.

have H : $\langle 0 < \text{lookup } x \ x' \implies (\lambda k. (\text{if } x' = k \text{ then } \text{Suc } 0 \text{ else } 0) + (\text{if } k = x' \wedge 0 < \text{lookup } x \ k \text{ then } \text{lookup } x \ k - 1 \text{ else } \text{lookup } x \ k)) = \text{lookup } x \ x' \rangle$

by auto

have [simp]: $\langle \text{finite } \{x. \ 0 < (\text{Suc } 0 \text{ when } x' = x)\} \rangle$ **for** $x' :: \text{nat}$ **and** x

by (smt bounded-nat-set-is-finite lessI mem-Collect-eq neq0-conv when-cong when-neq-zero)

have H : $\langle x' \in \text{keys } x \implies \text{monomial } (\text{Suc } 0) \ x' + \text{Abs-poly-mapping } (\lambda k. \text{if } k = x' \wedge 0 < \text{lookup } x \ k \text{ then } \text{lookup } x \ k - 1 \text{ else } \text{lookup } x \ k) = x \rangle$

for x **and** $x' :: \text{nat}$

apply (simp only: keys-def single.abs-eq)

apply (subst plus-poly-mapping.abs-eq)

by (auto simp: eq-onp-def when-def H)

intro!: finite-subset[of $\langle \{xa. (xa = x' \wedge 0 < \text{lookup } x \ xa \implies \text{Suc } 0 < \text{lookup } x \ x') \wedge (xa \neq x' \implies 0 < \text{lookup } x \ xa)\} \rangle \langle \{xa. \ 0 < \text{lookup } x \ xa\} \rangle$]

have [simp]: $\langle x' \in \text{keys } x \implies$

$\text{MPoly-Type.monom } (\text{monomial } (\text{Suc } 0) \ x' + \text{decrease-key } x' \ x) \ n =$

$\text{MPoly-Type.monom } x \ n \rangle$ **for** $x \ n$ **and** x'

apply (subst mpoly.mapping-of-inject[symmetric], subst poly-mapping.lookup-inject[symmetric])

unfolding mapping-of-monom lookup-single

apply (auto intro!: ext simp: decrease-key-def when-def H)

done

```

have  $pX$ :  $\langle ?pX = \text{monom} (\text{monomial} (\text{Suc } 0) x') 1 * (\sum x \in \{x \in ?I. x' \in \text{keys } x\}. \text{MPoly-Type.monom} (\text{decrease-key } x' x) (\text{MPoly-Type.coeff } p x)) \rangle$ 
  (is  $\langle - = - * ?pX' \rangle$ )
  by (subst sum-distrib-left, subst mult-monom)
  (auto intro!: sum.cong)
have  $\langle x' \notin \text{vars } ?qX \rangle$ 
  using vars-setsum[of  $\langle \{x. x \in \text{keys} (\text{mapping-of } p) \wedge x' \notin \text{keys } x \} \rangle \langle ?f \rangle$ ]
  by (auto dest!: vars-monom-subset[unfolded subset-eq Ball-def, rule-format])
then show  $?thesis$ 
  using that[of  $?pX' ?qX$ ]
  unfolding  $pX$ [symmetric] 1[symmetric]
  by blast
qed

```

lemma *polynomial-split-on-var2*:

```

fixes  $p$  ::  $\langle \text{int mpoly} \rangle$ 
assumes  $\langle x' \notin \text{vars } s \rangle$ 
obtains  $q r$  where
   $\langle p = (\text{monom} (\text{monomial} (\text{Suc } 0) x') 1 - s) * q + r \rangle$  and
   $\langle x' \notin \text{vars } r \rangle$ 
proof -
  have  $\text{eq}[simp]$ :  $\langle \text{monom} (\text{monomial} (\text{Suc } 0) x') 1 = \text{Var } x' \rangle$ 
    by (simp add: Var.abs-eq Var0-def monom.abs-eq)
  have  $\langle \forall m \leq n. \forall P :: \text{int mpoly}. \text{degree } P x' < m \implies (\exists A B. P = (\text{Var } x' - s) * A + B \wedge x' \notin \text{vars } B) \rangle$  for  $n$ 
  proof (induction n)
    case 0
    then show  $?case$  by auto
  next
    case (Suc n)
    then have  $IH$ :  $\langle m \leq n \implies \text{MPoly-Type.degree } P x' < m \implies (\exists A B. P = (\text{Var } x' - s) * A + B \wedge x' \notin \text{vars } B) \rangle$  for  $m P$ 
      by fast
    show  $?case$ 
  proof (intro allI impI)
    fix  $m$  and  $P$  ::  $\langle \text{int mpoly} \rangle$ 
    assume  $\langle m \leq \text{Suc } n \rangle$  and  $\text{deg}$ :  $\langle \text{MPoly-Type.degree } P x' < m \rangle$ 
    consider
       $\langle m \leq n \rangle$  |
       $\langle m = \text{Suc } n \rangle$ 
    using  $\langle m \leq \text{Suc } n \rangle$  by linarith
    then show  $\langle \exists A B. P = (\text{Var } x' - s) * A + B \wedge x' \notin \text{vars } B \rangle$ 
  proof cases
    case 1
    then show  $\langle ?thesis \rangle$ 
      using Suc deg by blast
  next
    case [simp]: 2
    obtain  $A B$  where
       $P$ :  $\langle P = \text{Var } x' * A + B \rangle$  and
       $\langle x' \notin \text{vars } B \rangle$ 
    using polynomial-split-on-var[of  $P x'$ ] unfolding eq by blast
    have  $P'$ :  $\langle P = (\text{Var } x' - s) * A + (s * A + B) \rangle$ 
      by (auto simp: field-simps P)

```

```

have  $\langle A = 0 \vee \text{degree } (s * A) \ x' < \text{degree } P \ x' \rangle$ 
  using  $\text{deg } \langle x' \notin \text{vars } B \rangle \ \langle x' \notin \text{vars } s \rangle \ \text{degree-times-le}[of \ s \ A \ x'] \ \text{deg}$ 
  unfolding  $P$ 
  by  $(\text{auto simp: degree-sum-notin degree-mult-Var' degree-mult-Var degree-notin-vars}$ 
     $\text{split: if-splits})$ 
then obtain  $A' \ B'$  where
   $sA: \langle s * A = (\text{Var } x' - s) * A' + B' \rangle$  and
   $\langle x' \notin \text{vars } B' \rangle$ 
  using  $IH[of \ \langle m-1 \rangle \ \langle s * A \rangle] \ \text{deg } \langle x' \notin \text{vars } B \rangle \ \text{that}[of \ 0 \ 0]$ 
  by  $(\text{cases } \langle 0 < n \rangle) \ (\text{auto dest!: vars-in-right-only})$ 
have  $\langle P = (\text{Var } x' - s) * (A + A') + (B' + B) \rangle$ 
  unfolding  $P' \ sA$  by  $(\text{auto simp: field-simps})$ 
moreover have  $\langle x' \notin \text{vars } (B' + B) \rangle$ 
  using  $\langle x' \notin \text{vars } B' \rangle \ \langle x' \notin \text{vars } B \rangle$ 
  by  $(\text{meson UnE subset-iff vars-add})$ 
ultimately show  $?thesis$ 
  by  $\text{fast}$ 
qed
qed
qed
then show  $?thesis$ 
  using  $\text{that unfolding eq}$ 
  by  $\text{blast}$ 
qed

lemma  $\text{finit-whenI}[intro]:$ 
 $\langle \text{finite } \{x. (0 :: \text{nat}) < (y \ x)\} \implies \text{finite } \{x. 0 < (y \ x \ \text{when } x \neq x')\} \rangle$ 
apply  $(\text{rule finite-subset})$ 
  defer apply  $\text{assumption}$ 
apply  $(\text{auto simp: when-def})$ 
done

lemma  $\text{polynomial-split-on-var-diff-sq2}:$ 
fixes  $p :: \langle \text{int mpoly} \rangle$ 
obtains  $q \ r \ s$  where
   $\langle p = \text{monom } (\text{monomial } (\text{Suc } 0) \ x') \ 1 * q + r + s * (\text{monom } (\text{monomial } (\text{Suc } 0) \ x') \ 1^{\wedge} 2 - \text{monom}$ 
     $(\text{monomial } (\text{Suc } 0) \ x') \ 1) \rangle$  and
   $\langle x' \notin \text{vars } r \rangle$  and
   $\langle x' \notin \text{vars } q \rangle$ 
proof  $-$ 
  let  $?v = \langle \text{monom } (\text{monomial } (\text{Suc } 0) \ x') \ 1 :: \text{int mpoly} \rangle$ 
  have  $H: \langle n < m \implies n > 0 \implies \exists q. ?v^{\wedge} n = ?v + q * (?v^{\wedge} 2 - ?v) \rangle$  for  $n \ m :: \text{nat}$ 
  proof  $(\text{induction } m \ \text{arbitrary: } n)$ 
    case  $0$ 
    then show  $?case$  by  $\text{auto}$ 
  next
    case  $(\text{Suc } m \ n)$  note  $IH = \text{this}(1-)$ 
    consider
     $\langle n < m \rangle \mid$ 
     $\langle m = n \rangle \ \langle n > 1 \rangle \mid$ 
     $\langle n = 1 \rangle$ 
    using  $IH$ 
    by  $(\text{cases } \langle n < m \rangle; \text{cases } n) \ \text{auto}$ 
  then show  $?case$ 
  proof  $\text{cases}$ 

```

```

case 1
then show ?thesis using IH by auto
next
case 2
have eq:  $\langle ?v^{(n)} = ((?v :: \text{int mpoly})^{(n-2)} * (?v^{(2)} - ?v) + ?v^{(n-1)}) \rangle$ 
  using 2 by (auto simp: field-simps power-eq-if
    ideal.scale-right-diff-distrib)
obtain q where
  q:  $\langle ?v^{(n-1)} = ?v + q * (?v^{(2)} - ?v) \rangle$ 
  using IH(1)[of <n-1>] 2
  by auto
show ?thesis
  using q unfolding eq
  by (auto intro!: exI[of - <?v^{(n-2)} + q>] simp: distrib-right)
next
case 3
then show <?thesis>
  by auto
qed
qed
have H:  $\langle n > 0 \implies \exists q. ?v^{(n)} = ?v + q * (?v^{(2)} - ?v) \rangle$  for n
  using H[of n <n+1>]
  by auto
obtain qr ::  $\langle \text{nat} \Rightarrow \text{int mpoly} \rangle$  where
  qr:  $\langle n > 0 \implies ?v^{(n)} = ?v + qr\ n * (?v^{(2)} - ?v) \rangle$  for n
  using H by metis
have vn:  $\langle (\text{if lookup } x\ x' = 0 \text{ then } 1 \text{ else } \text{Var } x' \wedge \text{lookup } x\ x') =$ 
   $(\text{if lookup } x\ x' = 0 \text{ then } 1 \text{ else } ?v) + (\text{if lookup } x\ x' = 0 \text{ then } 0 \text{ else } 1) * qr (\text{lookup } x\ x') * (?v^{(2)} - ?v) \rangle$ 
for x
  by (simp add: qr[symmetric] Var-def Var0-def monom.abs-eq[symmetric] cong: if-cong)

have q:  $\langle p = (\sum x \in \text{keys } (\text{mapping-of } p). \text{MPoly-Type.monom } x (\text{MPoly-Type.coeff } p\ x)) \rangle$ 
  by (rule polynomial-sum-monom(1)[of p])
have [simp]:
   $\langle \text{lookup } x\ x' = 0 \implies$ 
  Abs-poly-mapping  $(\lambda k. \text{lookup } x\ k \text{ when } k \neq x') = x \rangle$  for x
  by (cases x, auto simp: poly-mapping.Abs-poly-mapping-inject)
  (auto intro!: ext simp: when-def)
have [simp]:  $\langle \text{finite } \{x. 0 < (a \text{ when } x' = x)\} \rangle$  for a :: nat
  by (metis (no-types, lifting) infinite-nat-iff-unbounded less-not-refl lookup-single lookup-single-not-eq
mem-Collect-eq)

have [simp]:  $\langle ((\lambda x. x + \text{monomial } (\text{Suc } 0)\ x') \wedge (n))$ 
   $(\text{monomial } (\text{Suc } 0)\ x') = \text{Abs-poly-mapping } (\lambda k. (\text{if } k = x' \text{ then } n+1 \text{ else } 0)) \rangle$  for n
  by (induction n)
  (auto simp: single-def Abs-poly-mapping-inject plus-poly-mapping.abs-eq eq-onp-def cong:if-cong)
have [simp]:  $\langle 0 < \text{lookup } x\ x' \implies$ 
  Abs-poly-mapping  $(\lambda k. \text{lookup } x\ k \text{ when } k \neq x') +$ 
  Abs-poly-mapping  $(\lambda k. \text{if } k = x' \text{ then lookup } x\ x' - \text{Suc } 0 + 1 \text{ else } 0) =$ 
  x for x
apply (cases x, auto simp: poly-mapping.Abs-poly-mapping-inject plus-poly-mapping.abs-eq eq-onp-def)
apply (subst plus-poly-mapping.abs-eq)
apply (auto simp: poly-mapping.Abs-poly-mapping-inject plus-poly-mapping.abs-eq eq-onp-def)
apply (subst Abs-poly-mapping-inject)
apply auto

```

```

done
define f where
  ⟨f x = (MPoly-Type.monom (remove-key x' x) (MPoly-Type.coeff p x)) *
    (if lookup x x' = 0 then 1 else Var x' ^ (lookup x x'))⟩ for x
have f-alt-def: ⟨f x = MPoly-Type.monom x (MPoly-Type.coeff p x)⟩ for x
  by (auto simp: f-def monom-def remove-key-def Var-def MPoly-monomial-power Var0-def
    mpoly.MPoly-inject monomial-inj times-mpoly.abs-eq
    times-mpoly.abs-eq mult-single)
have p: ⟨p = (∑ x∈keys (mapping-of p).
  MPoly-Type.monom (remove-key x' x) (MPoly-Type.coeff p x) *
  (if lookup x x' = 0 then 1 else ?v)) +
  (∑ x∈keys (mapping-of p).
  MPoly-Type.monom (remove-key x' x) (MPoly-Type.coeff p x) *
  (if lookup x x' = 0 then 0
    else 1) * qr (lookup x x') *
  (?v2 - ?v))⟩
(is ⟨- = ?a + ?v2v⟩)
apply (subst q)
unfolding f-alt-def[symmetric, abs-def] f-def vn semiring-class.distrib-left
  comm-semiring-1-class.semiring-normalization-rules(18) semiring-0-class.sum-distrib-right
by (simp add: semiring-class.distrib-left
  sum.distrib)

have I: ⟨keys (mapping-of p) = {x ∈ keys (mapping-of p). lookup x x' = 0} ∪ {x ∈ keys (mapping-of
p). lookup x x' ≠ 0}⟩
by auto

have ⟨p = (∑ x | x ∈ keys (mapping-of p) ∧ lookup x x' = 0.
  MPoly-Type.monom x (MPoly-Type.coeff p x)) +
  (∑ x | x ∈ keys (mapping-of p) ∧ lookup x x' ≠ 0.
  MPoly-Type.monom (remove-key x' x) (MPoly-Type.coeff p x) *
  (MPoly-Type.monom (monomial (Suc 0) x') 1) +
  (∑ x | x ∈ keys (mapping-of p) ∧ lookup x x' ≠ 0.
  MPoly-Type.monom (remove-key x' x) (MPoly-Type.coeff p x) *
  qr (lookup x x') *
  (?v2 - ?v))⟩
(is ⟨p = ?A + ?B * - + ?C * -⟩)
unfolding semiring-0-class.sum-distrib-right[of - - ⟨(MPoly-Type.monom (monomial (Suc 0) x') 1)⟩]
apply (subst p)
apply (subst (2)I)
apply (subst I)
apply (subst comm-monoid-add-class.sum.union-disjoint)
apply auto[3]
apply (subst comm-monoid-add-class.sum.union-disjoint)
apply auto[3]
apply (subst (4) sum.cong[OF refl, of - - ⟨λx. MPoly-Type.monom (remove-key x' x) (MPoly-Type.coeff
p x) *
  qr (lookup x x')⟩])
apply (auto; fail)
apply (subst (3) sum.cong[OF refl, of - - ⟨λx. 0⟩])
apply (auto; fail)
apply (subst (2) sum.cong[OF refl, of - - ⟨λx. MPoly-Type.monom (remove-key x' x) (MPoly-Type.coeff
p x) *
  (MPoly-Type.monom (monomial (Suc 0) x') 1)⟩])
apply (auto; fail)

```

```

apply (subst (1) sum.cong[OF refl, of - - ⟨λx. MPoly-Type.monom x (MPoly-Type.coeff p x)⟩])
by (auto simp: f-def simp flip: f-alt-def)

moreover have ⟨x' ∉ vars ?A⟩
using vars-setsum[of ⟨{x ∈ keys (mapping-of p). lookup x x' = 0}⟩
  ⟨λx. MPoly-Type.monom x (MPoly-Type.coeff p x)⟩]
apply auto
apply (drule set-rev-mp, assumption)
apply (auto dest!: lookup-eq-zero-in-keys-contradict)
by (meson lookup-eq-zero-in-keys-contradict subsetD vars-monom-subset)
moreover have ⟨x' ∉ vars ?B⟩
using vars-setsum[of ⟨{x ∈ keys (mapping-of p). lookup x x' ≠ 0}⟩
  ⟨λx. MPoly-Type.monom (remove-key x' x) (MPoly-Type.coeff p x)⟩]
apply auto
apply (drule set-rev-mp, assumption)
apply (auto dest!: lookup-eq-zero-in-keys-contradict)
apply (drule subsetD[OF vars-monom-subset])
apply (auto simp: remove-key-keys[symmetric])
done
ultimately show ?thesis apply -
apply (rule that[of ?B ?A ?C])
apply (auto simp: ac-simps)
done
qed

```

```

lemma polynomial-decomp-alien-var:
fixes q A b :: ⟨int mpoly⟩
assumes
  q: ⟨q = A * (monom (monomial (Suc 0) x') 1) + b⟩ and
  x: ⟨x' ∉ vars q⟩ ⟨x' ∉ vars b⟩
shows
  ⟨A = 0⟩ and
  ⟨q = b⟩
proof -
let ?A = ⟨A * (monom (monomial (Suc 0) x') 1)⟩
have ⟨?A = q - b⟩
using arg-cong[OF q, of ⟨λa. a - b⟩]
by auto
moreover have ⟨x' ∉ vars (q - b)⟩
using x vars-in-right-only
by fastforce
ultimately have ⟨x' ∉ vars (?A)⟩
by simp
then have ⟨?A = 0⟩
by (auto simp: vars-mult-monom split: if-splits)
moreover have ⟨?A = 0 ⟹ A = 0⟩
by (metis empty-not-insert mult-zero-left vars-mult-monom)
ultimately show ⟨A = 0⟩
by blast
then show ⟨q = b⟩
using q by auto
qed

```

```

lemma polynomial-decomp-alien-var2:

```

fixes $q A b :: \langle \text{int mpoly} \rangle$
assumes
 $q: \langle q = A * (\text{monom} (\text{monomial} (\text{Suc } 0) x') 1 + p) + b \rangle$ **and**
 $x: \langle x' \notin \text{vars } q \rangle \langle x' \notin \text{vars } b \rangle \langle x' \notin \text{vars } p \rangle$
shows
 $\langle A = 0 \rangle$ **and**
 $\langle q = b \rangle$
proof –
let $?x = \langle \text{monom} (\text{monomial} (\text{Suc } 0) x') 1 \rangle$
have $x'[\text{simp}] : \langle ?x = \text{Var } x' \rangle$
by (*simp add: Var.abs-eq Var₀-def monom.abs-eq*)
have $\langle \exists n Ax A'. A = ?x * Ax + A' \wedge x' \notin \text{vars } A' \wedge \text{degree } Ax x' = n \rangle$
using *polynomial-split-on-var*[of $A x'$] **by** *metis*
from *wellorder-class.exists-least-iff*[*THEN iffD1, OF this*] **obtain** $Ax A' n$ **where**
 $A: \langle A = Ax * ?x + A' \rangle$ **and**
 $\langle x' \notin \text{vars } A' \rangle$ **and**
 $n: \langle \text{MPoly-Type.degree } Ax x' = n \rangle$ **and**
 $H: \langle \bigwedge m Ax A'. m < n \longrightarrow$
 $A \neq Ax * \text{MPoly-Type.monom} (\text{monomial} (\text{Suc } 0) x') 1 + A' \vee$
 $x' \in \text{vars } A' \vee \text{MPoly-Type.degree } Ax x' \neq m \rangle$
unfolding *wellorder-class.exists-least-iff*[of $\langle \lambda n. \exists Ax A'. A = Ax * ?x + A' \wedge x' \notin \text{vars } A' \wedge$
 $\text{degree } Ax x' = n \rangle$]
by (*auto simp: field-simps*)

have $\langle q = (A + Ax * p) * \text{monom} (\text{monomial} (\text{Suc } 0) x') 1 + (p * A' + b) \rangle$
unfolding $q A$ **by** (*auto simp: field-simps*)
moreover have $\langle x' \notin \text{vars } q \rangle \langle x' \notin \text{vars } (p * A' + b) \rangle$
using $x \langle x' \notin \text{vars } A' \rangle$
by (*smt UnE add.assoc add.commute calculation subset-iff vars-in-right-only vars-mult*)
ultimately have $\langle A + Ax * p = 0 \rangle \langle q = p * A' + b \rangle$
by (*rule polynomial-decomp-alien-var*)

have $A': \langle A' = -Ax * ?x - Ax * p \rangle$
using $\langle A + Ax * p = 0 \rangle$ **unfolding** A
by (*metis (no-types, lifting) add-uminus-conv-diff eq-neg-iff-add-eq-0 minus-add-cancel*
 mult-minus-left)

have $\langle A = -(Ax * p) \rangle$
using A **unfolding** A'
apply *auto*
done

obtain $Axx Ax'$ **where**
 $Ax: \langle Ax = ?x * Axx + Ax' \rangle$ **and**
 $\langle x' \notin \text{vars } Ax' \rangle$
using *polynomial-split-on-var*[of $Ax x'$] **by** *metis*

have $\langle A = ?x * (-Axx * p) + (-Ax' * p) \rangle$
unfolding $\langle A = -(Ax * p) \rangle Ax$
by (*auto simp: field-simps*)

moreover have $\langle x' \notin \text{vars } (-Ax' * p) \rangle$
using $\langle x' \notin \text{vars } Ax' \rangle$ **by** (*metis (no-types, opaque-lifting) UnE add.right-neutral*
 $\text{add-minus-cancel assms(4) subsetD vars-in-right-only vars-mult}$)
moreover have $\langle Axx \neq 0 \implies \text{MPoly-Type.degree } (-Axx * p) x' < \text{degree } Ax x' \rangle$

using *degree-times-le*[of $Axx\ p\ x'$] x
by (*auto simp*: $Ax\ \text{degree-sum-notin}\ \langle x' \notin \text{vars}\ Ax' \rangle\ \text{degree-mult-Var}'$
degree-notin-vars)
ultimately have [*simp*]: $\langle Axx = 0 \rangle$
using H [of $\langle MPoly\text{-Type.degree}\ (-\ Axx * p)\ x' \rangle\ \langle -\ Axx * p \rangle\ \langle -\ Ax' * p \rangle$]
by (*auto simp*: n)
then have [*simp*]: $\langle Ax' = Ax \rangle$
using Ax **by** *auto*

show $\langle A = 0 \rangle$
using $A\ \langle A = -\ (Ax * p) \rangle\ \langle x' \notin \text{vars}\ (-\ Ax' * p) \rangle\ \langle x' \notin \text{vars}\ A' \rangle\ \text{polynomial-decomp-alien-var}(1)$
by *force*
then show $\langle q = b \rangle$
using q **by** *auto*
qed

lemma *vars-unE*: $\langle x \in \text{vars}\ (a * b) \implies (x \in \text{vars}\ a \implies \text{thesis}) \implies (x \in \text{vars}\ b \implies \text{thesis}) \implies \text{thesis} \rangle$
using *vars-mult*[of $a\ b$] **by** *auto*

lemma *in-keys-minusI1*:
assumes $t \in \text{keys}\ p$ **and** $t \notin \text{keys}\ q$
shows $t \in \text{keys}\ (p - q)$
using *assms* **unfolding** *in-keys-iff lookup-minus* **by** *simp*

lemma *in-keys-minusI2*:
fixes $t :: \langle 'a \rangle$ **and** $q :: \langle 'a \Rightarrow_0 'b :: \{\text{cancel-comm-monoid-add,group-add}\} \rangle$
assumes $t \in \text{keys}\ q$ **and** $t \notin \text{keys}\ p$
shows $t \in \text{keys}\ (p - q)$
using *assms* **unfolding** *in-keys-iff lookup-minus* **by** *simp*

lemma *in-vars-addE*:
 $\langle x \in \text{vars}\ (p + q) \implies (x \in \text{vars}\ p \implies \text{thesis}) \implies (x \in \text{vars}\ q \implies \text{thesis}) \implies \text{thesis} \rangle$
by (*meson UnE in-mono vars-add*)

lemma *lookup-monomial-If*:
 $\langle \text{lookup}\ (\text{monomial}\ v\ k) = (\lambda k'. \text{if } k = k' \text{ then } v \text{ else } 0) \rangle$
by (*intro ext*) (*auto simp*: *lookup-single-not-eq*)

lemma *vars-mult-Var*:
 $\langle \text{vars}\ (\text{Var}\ x * p) = (\text{if } p = 0 \text{ then } \{\} \text{ else insert } x\ (\text{vars}\ p)) \rangle$ **for** $p :: \langle \text{int mpoly} \rangle$

proof –

have $\langle p \neq 0 \implies$

$\exists xa. (\exists k. xa = \text{monomial}\ (\text{Suc}\ 0)\ x + k) \wedge$

$\text{lookup}\ (\text{mapping-of}\ p)\ (xa - \text{monomial}\ (\text{Suc}\ 0)\ x) \neq 0 \wedge$

$0 < \text{lookup}\ xa\ x \rangle$

by (*metis* (*no-types*, *opaque-lifting*) *One-nat-def ab-semigroup-add-class.add commute*
add-diff-cancel-right' aux lookup-add lookup-single-eq mapping-of-inject
neq0-conv one-neq-zero plus-eq-zero-2 zero-mpoly.rep-eq)

then show *?thesis*

apply (*auto simp*: *vars-def times-mpoly.rep-eq Var.rep-eq*

elim!: *in-keys-timesE dest: keys-add'*)

apply (*auto simp*: *keys-def lookup-times-monomial-left Var.rep-eq Var₀-def adds-def*

lookup-add eq-diff-eq[*symmetric*])

done
qed

lemma *keys-mult-monomial*:

$\langle \text{keys } (\text{monomial } (n :: \text{int}) \ k \ * \ \text{mapping-of } a) = (\text{if } n = 0 \ \text{then } \{\} \ \text{else } ((+) \ k) \ \text{'keys } (\text{mapping-of } a)) \rangle$

proof –

have [*simp*]: $\langle (\sum aa. (\text{if } k = aa \ \text{then } n \ \text{else } 0) \ * \ (\sum q. \text{lookup } (\text{mapping-of } a) \ q \ \text{when } k + xa = aa + q)) = (\sum aa. (\text{if } k = aa \ \text{then } n \ * \ (\sum q. \text{lookup } (\text{mapping-of } a) \ q \ \text{when } k + xa = aa + q) \ \text{else } 0)) \rangle$

for *xa*

by (*smt Sum-any.cong mult-not-zero*)

show *?thesis*

apply (*auto simp: vars-def times-mpoly.rep-eq Const.rep-eq times-poly-mapping.rep-eq Const₀-def elim!: in-keys-timesE split: if-splits*)

apply (*auto simp: lookup-monomial-If prod-fun-def keys-def times-poly-mapping.rep-eq*)

done

qed

lemma *vars-mult-Const*:

$\langle \text{vars } (\text{Const } n \ * \ a) = (\text{if } n = 0 \ \text{then } \{\} \ \text{else } \text{vars } a) \rangle$ **for** *a* :: $\langle \text{int mpoly} \rangle$

by (*auto simp: vars-def times-mpoly.rep-eq Const.rep-eq keys-mult-monomial Const₀-def elim!: in-keys-timesE split: if-splits*)

lemma *coeff-minus*: $\text{coeff } p \ m - \text{coeff } q \ m = \text{coeff } (p - q) \ m$

by (*simp add: coeff-def lookup-minus minus-mpoly.rep-eq*)

lemma *Const-1-eq-1*: $\langle \text{Const } (1 :: \text{int}) = (1 :: \text{int mpoly}) \rangle$

by (*simp add: Const.abs-eq Const₀-one one-mpoly.abs-eq*)

lemma [*simp*]:

$\langle \text{vars } (1 :: \text{int mpoly}) = \{\} \rangle$

by (*auto simp: vars-def one-mpoly.rep-eq Const-1-eq-1*)

2.2 More Ideals

lemma

fixes *A* :: $\langle (('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a :: \text{comm-ring-1}) \ \text{set} \rangle$

assumes $\langle p \in \text{ideal } A \rangle$

shows $\langle p * q \in \text{ideal } A \rangle$

by (*metis assms ideal.span-scale semiring-normalization-rules(7)*)

The following theorem is very close to *More-Modules.ideal (insert ?a ?S) = {x. $\exists k. x - k * ?a \in \text{More-Modules.ideal } ?S$ }*, except that it is more useful if we need to take an element of *More-Modules.ideal (insert a S)*.

lemma *ideal-insert'*:

$\langle \text{More-Modules.ideal } (\text{insert } a \ S) = \{y. \exists x \ k. y = x + k * a \wedge x \in \text{More-Modules.ideal } S\} \rangle$

apply (*auto simp: ideal.span-insert*

*intro: exI[of - $\langle - - k * a \rangle$])*

apply (*rule-tac x = $\langle x - k * a \rangle$ in exI*)

apply *auto*

apply (*rule-tac x = $\langle k \rangle$ in exI*)

apply *auto*

done

lemma *ideal-mult-right-in*:

$\langle a \in \text{ideal } A \implies a * b \in \text{More-Modules.ideal } A \rangle$
by (*metis ideal.span-scale mult.commute*)

lemma *ideal-mult-right-in2*:

$\langle a \in \text{ideal } A \implies b * a \in \text{More-Modules.ideal } A \rangle$
by (*metis ideal.span-scale*)

lemma [*simp*]: $\langle \text{vars } (\text{Var } x :: 'a :: \{\text{zero-neq-one}\} \text{mpoly}) = \{x\} \rangle$

by (*auto simp: vars-def Var.rep-eq Var₀-def*)

lemma *vars-minus-Var-subset*:

$\langle \text{vars } (p' - \text{Var } x :: 'a :: \{\text{ab-group-add,one,zero-neq-one}\} \text{mpoly}) \subseteq \mathcal{V} \implies \text{vars } p' \subseteq \text{insert } x \mathcal{V} \rangle$
using *vars-add*[of $\langle p' - \text{Var } x \rangle \langle \text{Var } x \rangle$]
by *auto*

lemma *vars-add-Var-subset*:

$\langle \text{vars } (p' + \text{Var } x :: 'a :: \{\text{ab-group-add,one,zero-neq-one}\} \text{mpoly}) \subseteq \mathcal{V} \implies \text{vars } p' \subseteq \text{insert } x \mathcal{V} \rangle$
using *vars-add*[of $\langle p' + \text{Var } x \rangle \langle - \text{Var } x \rangle$]
by *auto*

lemma *coeff-monomila-in-varsD*:

$\langle \text{coeff } p (\text{monomial } (\text{Suc } 0) x) \neq 0 \implies x \in \text{vars } (p :: \text{int mpoly}) \rangle$
by (*auto simp: coeff-def vars-def keys-def*
intro!: exI[of $\langle \text{monomial } (\text{Suc } 0) x \rangle$])

lemma *coeff-MPoly-monomial*[*simp*]:

$\langle (\text{MPoly-Type.coeff } (\text{MPoly } (\text{monomial } a m)) m) = a \rangle$
by (*metis MPoly-Type.coeff-def lookup-single-eq monom.abs-eq monom.rep-eq*)

end

theory *Finite-Map-Multiset*

imports

HOL-Library.Finite-Map

Nested-Multisets-Ordinals.Duplicate-Free-Multiset

begin

notation *image-mset* (**infixr** $\#$ 90)

3 Finite maps and multisets

3.1 Finite sets and multisets

abbreviation *mset-fset* :: $\langle 'a \text{ fset} \Rightarrow 'a \text{ multiset} \rangle$ **where**

$\langle \text{mset-fset } N \equiv \text{mset-set } (\text{fset } N) \rangle$

definition *fset-mset* :: $\langle 'a \text{ multiset} \Rightarrow 'a \text{ fset} \rangle$ **where**

$\langle \text{fset-mset } N \equiv \text{Abs-fset } (\text{set-mset } N) \rangle$

lemma *fset-mset-mset-fset*: $\langle \text{fset-mset } (\text{mset-fset } N) = N \rangle$

by (*auto simp: fset.fset-inverse fset-mset-def*)

lemma *mset-fset-fset-mset*[*simp*]:

$\langle \text{mset-fset } (fset\text{-mset } N) = \text{remdups-mset } N \rangle$
by (auto simp: fset.fset-inverse fset-mset-def Abs-fset-inverse remdups-mset-def)

lemma *in-mset-fset-fmmember*[simp]: $\langle x \in\# \text{mset-fset } N \longleftrightarrow x \in| N \rangle$
by (auto simp: fmmember.rep-eq)

lemma *in-fset-mset-mset*[simp]: $\langle x \in| fset\text{-mset } N \longleftrightarrow x \in\# N \rangle$
by (auto simp: fmmember.rep-eq fset-mset-def Abs-fset-inverse)

3.2 Finite map and multisets

Roughly the same as *ran* and *dom*, but with duplication in the content (unlike their finite sets counterpart) while still working on finite domains (unlike a function mapping). Remark that *dom-m* (the keys) does not contain duplicates, but we keep for symmetry (and for easier use of multiset operators as in the definition of *ran-m*).

definition *dom-m where*
 $\langle \text{dom-m } N = \text{mset-fset } (f\text{mdom } N) \rangle$

definition *ran-m where*
 $\langle \text{ran-m } N = \text{the } \# \text{fmlookup } N \text{ } \# \text{dom-m } N \rangle$

lemma *dom-m-fmdrop*[simp]: $\langle \text{dom-m } (f\text{mdrop } C N) = \text{remove1-mset } C (\text{dom-m } N) \rangle$
unfolding *dom-m-def*
by (cases $\langle C \in| f\text{mdom } N \rangle$)
(auto simp: mset-set.remove fmmember.rep-eq)

lemma *dom-m-fmdrop-All*: $\langle \text{dom-m } (f\text{mdrop } C N) = \text{removeAll-mset } C (\text{dom-m } N) \rangle$
unfolding *dom-m-def*
by (cases $\langle C \in| f\text{mdom } N \rangle$)
(auto simp: mset-set.remove fmmember.rep-eq)

lemma *dom-m-fmupd*[simp]: $\langle \text{dom-m } (f\text{mupd } k C N) = \text{add-mset } k (\text{remove1-mset } k (\text{dom-m } N)) \rangle$
unfolding *dom-m-def*
by (cases $\langle k \in| f\text{mdom } N \rangle$)
(auto simp: mset-set.remove fmmember.rep-eq mset-set.insert-remove)

lemma *distinct-mset-dom*: $\langle \text{distinct-mset } (\text{dom-m } N) \rangle$
by (simp add: distinct-mset-mset-set dom-m-def)

lemma *in-dom-m-lookup-iff*: $\langle C \in\# \text{dom-m } N' \longleftrightarrow \text{fmlookup } N' C \neq \text{None} \rangle$
by (auto simp: dom-m-def fmdom.rep-eq fmlookup-dom'-iff)

lemma *in-dom-in-ran-m*[simp]: $\langle i \in\# \text{dom-m } N \implies \text{the } (f\text{mlookup } N i) \in\# \text{ran-m } N \rangle$
by (auto simp: ran-m-def)

lemma *fmupd-same*[simp]:
 $\langle x1 \in\# \text{dom-m } x1aa \implies f\text{mupd } x1 (\text{the } (f\text{mlookup } x1aa x1)) x1aa = x1aa \rangle$
by (metis fmap-ext fmupd-lookup in-dom-m-lookup-iff option.collapse)

lemma *ran-m-fmempty*[simp]: $\langle \text{ran-m } f\text{mempty} = \{ \# \} \rangle$ **and**
 dom-m-fmempty [simp]: $\langle \text{dom-m } f\text{mempty} = \{ \# \} \rangle$
by (auto simp: ran-m-def dom-m-def)

lemma *fmrestrict-set-fmupd*:
 $\langle a \in xs \implies f\text{mrestrict-set } xs (f\text{mupd } a C N) = f\text{mupd } a C (f\text{mrestrict-set } xs N) \rangle$

$\langle a \notin xs \implies \text{fmrestrict-set } xs \text{ (fmupd } a \ C \ N) = \text{fmrestrict-set } xs \ N \rangle$
by (auto simp: fmfilter-alt-defs)

lemma fset-fmdom-fmrestrict-set:
 $\langle \text{fset (fmdom (fmrestrict-set } xs \ N)) = \text{fset (fmdom } N) \cap xs \rangle$
by (auto simp: fmfilter-alt-defs)

lemma dom-m-fmrestrict-set: $\langle \text{dom-m (fmrestrict-set (set } xs) \ N) = \text{mset } xs \cap \# \text{ dom-m } N \rangle$
using fset-fmdom-fmrestrict-set[of $\langle \text{set } xs \rangle \ N$] distinct-mset-dom[of N]
distinct-mset-inter-remdups-mset[of $\langle \text{mset-fset (fmdom } N) \rangle \ \langle \text{mset } xs \rangle]$
by (auto simp: dom-m-def fset-mset-mset-fset finite-mset-set-inter multiset-inter-commute
remdups-mset-def)

lemma dom-m-fmrestrict-set': $\langle \text{dom-m (fmrestrict-set } xs \ N) = \text{mset-set } (xs \cap \text{set-mset (dom-m } N)) \rangle$
using fset-fmdom-fmrestrict-set[of $\langle xs \rangle \ N$] distinct-mset-dom[of N]
by (auto simp: dom-m-def fset-mset-mset-fset finite-mset-set-inter multiset-inter-commute
remdups-mset-def)

lemma indom-mI: $\langle \text{fmlookup } m \ x = \text{Some } y \implies x \in \# \text{ dom-m } m \rangle$
by (drule fmdomI) (auto simp: dom-m-def fmlookup.rep-eq)

lemma fmupd-fmdrop-id:
assumes $\langle k \in \# \text{ fmdom } N' \rangle$
shows $\langle \text{fmupd } k \text{ (the (fmlookup } N' \ k)) \text{ (fmdrop } k \ N') = N' \rangle$

proof –

have [simp]: $\langle \text{map-upd } k \text{ (the (fmlookup } N' \ k)) \text{ (}\lambda x. \text{ if } x \neq k \text{ then fmlookup } N' \ x \text{ else None)} = \text{map-upd } k \text{ (the (fmlookup } N' \ k)) \text{ (fmlookup } N') \rangle$

by (auto intro!: ext simp: map-upd-def)

have [simp]: $\langle \text{map-upd } k \text{ (the (fmlookup } N' \ k)) \text{ (fmlookup } N') = \text{fmlookup } N' \rangle$
using assms

by (auto intro!: ext simp: map-upd-def)

have [simp]: $\langle \text{finite (dom (}\lambda x. \text{ if } x = k \text{ then None else fmlookup } N' \ x)) \rangle$
by (subst dom-if) auto

show ?thesis

apply (auto simp: fmupd-def fmupd.abs-eq[symmetric])

unfolding fmdrop

apply (simp add: fmlookup-inverse)

done

qed

lemma fm-member-split: $\langle k \in \# \text{ fmdom } N' \implies \exists N'' \ v. N' = \text{fmupd } k \ v \ N'' \wedge \text{the (fmlookup } N' \ k) = v \wedge k \notin \# \text{ fmdom } N'' \rangle$
by (rule exI[of - $\langle \text{fmdrop } k \ N' \rangle$])
(auto simp: fmupd-fmdrop-id)

lemma $\langle \text{fmdrop } k \text{ (fmupd } k \ va \ N'') = \text{fmdrop } k \ N'' \rangle$
by (simp add: fmap-ext)

lemma fmap-ext-fmdom:
 $\langle \text{fmdom } N = \text{fmdom } N' \implies (\bigwedge x. x \in \# \text{ fmdom } N \implies \text{fmlookup } N \ x = \text{fmlookup } N' \ x) \implies N = N' \rangle$
by (rule fmap-ext)

(*case-tac* $\langle x \in | \text{fmdom } N \rangle$, *auto simp: fmdom-notD*)

lemma *fmrestrict-set-insert-in:*

$\langle xa \in \text{fset } (\text{fmdom } N) \implies$
 $\text{fmrestrict-set } (\text{insert } xa \text{ } l1) N = \text{fmupd } xa \text{ } (\text{the } (\text{fmlookup } N \text{ } xa)) \text{ } (\text{fmrestrict-set } l1 N) \rangle$
apply (*rule fmap-ext-fmdom*)
apply (*auto simp: fset-fmdom-fmrestrict-set fmember.rep-eq notin-fset; fail*)
apply (*auto simp: fmlookup-dom-iff; fail*)
done

lemma *fmrestrict-set-insert-notin:*

$\langle xa \notin \text{fset } (\text{fmdom } N) \implies$
 $\text{fmrestrict-set } (\text{insert } xa \text{ } l1) N = \text{fmrestrict-set } l1 N \rangle$
by (*rule fmap-ext-fmdom*)
(*auto simp: fset-fmdom-fmrestrict-set fmember.rep-eq notin-fset*)

lemma *fmrestrict-set-insert-in-dom-m[simp]:*

$\langle xa \in \# \text{ dom-m } N \implies$
 $\text{fmrestrict-set } (\text{insert } xa \text{ } l1) N = \text{fmupd } xa \text{ } (\text{the } (\text{fmlookup } N \text{ } xa)) \text{ } (\text{fmrestrict-set } l1 N) \rangle$
by (*simp add: fmrestrict-set-insert-in dom-m-def*)

lemma *fmrestrict-set-insert-notin-dom-m[simp]:*

$\langle xa \notin \# \text{ dom-m } N \implies$
 $\text{fmrestrict-set } (\text{insert } xa \text{ } l1) N = \text{fmrestrict-set } l1 N \rangle$
by (*simp add: fmrestrict-set-insert-notin dom-m-def*)

lemma *fmlookup-restrict-set-id:* $\langle \text{fset } (\text{fmdom } N) \subseteq A \implies \text{fmrestrict-set } A \text{ } N = N \rangle$

by (*metis fmap-ext fmdom'-alt-def fmdom'-notD fmlookup-restrict-set subset-iff*)

lemma *fmlookup-restrict-set-id':* $\langle \text{set-mset } (\text{dom-m } N) \subseteq A \implies \text{fmrestrict-set } A \text{ } N = N \rangle$

by (*rule fmlookup-restrict-set-id*)
(*auto simp: dom-m-def*)

lemma *ran-m-mapsto-upd:*

assumes

NC: $\langle C \in \# \text{ dom-m } N \rangle$

shows $\langle \text{ran-m } (\text{fmupd } C \text{ } C' \text{ } N) = \text{add-mset } C' \text{ } (\text{remove1-mset } (\text{the } (\text{fmlookup } N \text{ } C)) \text{ } (\text{ran-m } N)) \rangle$

proof –

define *N'* **where**

$\langle N' = \text{fmdrop } C \text{ } N \rangle$

have *N-N'*: $\langle \text{dom-m } N = \text{add-mset } C \text{ } (\text{dom-m } N') \rangle$

using *NC* **unfolding** *N'-def* **by** *auto*

have $\langle C \notin \# \text{ dom-m } N' \rangle$

using *NC* *distinct-mset-dom*[*of N*] **unfolding** *N-N'* **by** *auto*

then show *?thesis*

by (*auto simp: N-N' ran-m-def mset-set.insert-remove image-mset-remove1-mset-if*
intro!: image-mset-cong)

qed

lemma *ran-m-mapsto-upd-notin:*

assumes *NC:* $\langle C \notin \# \text{ dom-m } N \rangle$

shows $\langle \text{ran-m } (\text{fmupd } C \text{ } C' \text{ } N) = \text{add-mset } C' \text{ } (\text{ran-m } N) \rangle$

using *NC*

by (*auto simp: ran-m-def mset-set.insert-remove image-mset-remove1-mset-if*
intro!: image-mset-cong split: if-splits)

lemma *image-mset-If-eq-notin*:

$\langle C \notin\# A \implies \{\#f \text{ (if } x = C \text{ then } a \text{ } x \text{ else } b \text{ } x). x \in\# A \#\} = \{\#f(b \text{ } x). x \in\# A \#\} \rangle$
by (*induction A*) *auto*

lemma *filter-mset-cong2*:

$\langle \bigwedge x. x \in\# M \implies f \text{ } x = g \text{ } x \implies M = N \implies \text{filter-mset } f \text{ } M = \text{filter-mset } g \text{ } N \rangle$
by (*hypsubst, rule filter-mset-cong, simp*)

lemma *ran-m-fmdrop*:

$\langle C \in\# \text{dom-m } N \implies \text{ran-m (fmdrop } C \text{ } N) = \text{remove1-mset (the (fmlookup } N \text{ } C)) (ran-m } N) \rangle$
using *distinct-mset-dom[of N]*
by (*cases (fmlookup N C)*)
(auto simp: ran-m-def image-mset-If-eq-notin[of C - (λx. fst (the x))]
dest!: multi-member-split
intro!: filter-mset-cong2 image-mset-cong2)

lemma *ran-m-fmdrop-notin*:

$\langle C \notin\# \text{dom-m } N \implies \text{ran-m (fmdrop } C \text{ } N) = \text{ran-m } N \rangle$
using *distinct-mset-dom[of N]*
by (*auto simp: ran-m-def image-mset-If-eq-notin[of C - (λx. fst (the x))]*
dest!: multi-member-split
intro!: filter-mset-cong2 image-mset-cong2)

lemma *ran-m-fmdrop-If*:

$\langle \text{ran-m (fmdrop } C \text{ } N) = (\text{if } C \in\# \text{dom-m } N \text{ then remove1-mset (the (fmlookup } N \text{ } C)) (ran-m } N) \text{ else } \text{ran-m } N) \rangle$
using *distinct-mset-dom[of N]*
by (*auto simp: ran-m-def image-mset-If-eq-notin[of C - (λx. fst (the x))]*
dest!: multi-member-split
intro!: filter-mset-cong2 image-mset-cong2)

lemma *dom-m-empty-iff[iff]*:

$\langle \text{dom-m } NU = \{\#\} \iff NU = \text{fmempty} \rangle$
by (*cases NU*) (*auto simp: dom-m-def mset-set.insert-remove*)

end

theory *WB-Sort*

imports

Refine-Imperative-HOL.IICF

HOL-Library.Rewrite

Nested-Multisets-Ordinals.Duplicate-Free-Multiset

begin

This a complete copy-paste of the IsaFoL version because sharing is too hard.

Every element between *lo* and *hi* can be chosen as pivot element.

definition *choose-pivot* :: $\langle ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \text{ list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat nres} \rangle$ **where**
 $\langle \text{choose-pivot } - - - \text{lo } hi = \text{SPEC}(\lambda k. k \geq lo \wedge k \leq hi) \rangle$

The element at index *p* partitions the subarray *lo..hi*. This means that every element

definition *isPartition-wrt* :: $\langle ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow 'b \text{ list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{isPartition-wrt } R \text{ } xs \text{ } lo \text{ } hi \text{ } p \equiv (\forall i. i \geq lo \wedge i < p \longrightarrow R (xs!i) (xs!p)) \wedge (\forall j. j > p \wedge j \leq hi \longrightarrow R (xs!p) (xs!j)) \rangle$

lemma *isPartition-wrtI*:

$\langle (\bigwedge i. \llbracket i \geq lo; i < p \rrbracket \implies R (xs!i) (xs!p)) \implies (\bigwedge j. \llbracket j > p; j \leq hi \rrbracket \implies R (xs!p) (xs!j)) \implies isPartition-wrt R xs lo hi p \rangle$

by (*simp add: isPartition-wrt-def*)

definition *isPartition* :: $\langle 'a :: order\ list \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow bool \rangle$ **where**

$\langle isPartition\ xs\ lo\ hi\ p \equiv isPartition-wrt\ (\leq)\ xs\ lo\ hi\ p \rangle$

abbreviation *isPartition-map* :: $\langle ('b \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a\ list \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow bool \rangle$
where

$\langle isPartition-map\ R\ h\ xs\ i\ j\ k \equiv isPartition-wrt\ (\lambda a\ b. R\ (h\ a)\ (h\ b))\ xs\ i\ j\ k \rangle$

lemma *isPartition-map-def'*:

$\langle lo \leq p \implies p \leq hi \implies hi < length\ xs \implies isPartition-map\ R\ h\ xs\ lo\ hi\ p = isPartition-wrt\ R\ (map\ h\ xs)\ lo\ hi\ p \rangle$

by (*auto simp add: isPartition-wrt-def conjI*)

Example: 6 is the pivot element (with index 4); $7::'a$ is equal to the *length xs - 1*.

lemma $\langle isPartition\ [0,5,3,4,6,9,8,10::nat]\ 0\ 7\ 4 \rangle$

by (*auto simp add: isPartition-def isPartition-wrt-def nth-Cons'*)

definition *sublist* :: $\langle 'a\ list \Rightarrow nat \Rightarrow nat \Rightarrow 'a\ list \rangle$ **where**

$\langle sublist\ xs\ i\ j \equiv take\ (Suc\ j - i)\ (drop\ i\ xs) \rangle$

lemma *take-Suc0*:

$l \neq [] \implies take\ (Suc\ 0)\ l = [!0]$

$0 < length\ l \implies take\ (Suc\ 0)\ l = [!0]$

$Suc\ n \leq length\ l \implies take\ (Suc\ 0)\ l = [!0]$

by (*cases l, auto*)⁺

lemma *sublist-single*: $\langle i < length\ xs \implies sublist\ xs\ i\ i = [xs!i] \rangle$

by (*cases xs*) (*auto simp add: sublist-def take-Suc0*)

lemma *insert-eq*: $\langle insert\ a\ b = b \cup \{a\} \rangle$

by *auto*

lemma *sublist-nth*: $\langle \llbracket lo \leq hi; hi < length\ xs; k+lo \leq hi \rrbracket \implies (sublist\ xs\ lo\ hi)!k = xs!(lo+k) \rangle$

by (*simp add: sublist-def*)

lemma *sublist-length*: $\langle \llbracket i \leq j; j < length\ xs \rrbracket \implies length\ (sublist\ xs\ i\ j) = 1 + j - i \rangle$

by (*simp add: sublist-def*)

lemma *sublist-not-empty*: $\langle \llbracket i \leq j; j < length\ xs; xs \neq [] \rrbracket \implies sublist\ xs\ i\ j \neq [] \rangle$

apply *simp*

apply (*rewrite List.length-greater-0-conv[symmetric]*)

apply (*rewrite sublist-length*)

by *auto*

lemma *sublist-app*: $\langle \llbracket i1 \leq i2; i2 \leq i3 \rrbracket \implies sublist\ xs\ i1\ i2 @ sublist\ xs\ (Suc\ i2)\ i3 = sublist\ xs\ i1\ i3 \rangle$

unfolding *sublist-def*
by (*smt Suc-eq-plus1-left Suc-le-mono append.assoc le-SucI le-add-diff-inverse le-trans same-append-eq take-add*)

definition *sorted-sublist-wrt* :: $\langle 'b \Rightarrow 'b \Rightarrow \text{bool} \rangle \Rightarrow 'b \text{ list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{sorted-sublist-wrt } R \text{ } xs \text{ } lo \text{ } hi = \text{sorted-wrt } R \text{ } (\text{sublist } xs \text{ } lo \text{ } hi) \rangle$

definition *sorted-sublist* :: $\langle 'a :: \text{linorder list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{sorted-sublist } xs \text{ } lo \text{ } hi = \text{sorted-sublist-wrt } (\leq) \text{ } xs \text{ } lo \text{ } hi \rangle$

abbreviation *sorted-sublist-map* :: $\langle 'b \Rightarrow 'b \Rightarrow \text{bool} \rangle \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \text{ list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool} \rangle$
where
 $\langle \text{sorted-sublist-map } R \text{ } h \text{ } xs \text{ } lo \text{ } hi \equiv \text{sorted-sublist-wrt } (\lambda a \text{ } b. R \text{ } (h \text{ } a) \text{ } (h \text{ } b)) \text{ } xs \text{ } lo \text{ } hi \rangle$

lemma *sorted-sublist-map-def'*:
 $\langle lo < \text{length } xs \implies \text{sorted-sublist-map } R \text{ } h \text{ } xs \text{ } lo \text{ } hi \equiv \text{sorted-sublist-wrt } R \text{ } (\text{map } h \text{ } xs) \text{ } lo \text{ } hi \rangle$
apply (*simp add: sorted-sublist-wrt-def*)
by (*simp add: drop-map sorted-wrt-map sublist-def take-map*)

lemma *sorted-sublist-wrt-refl*: $\langle i < \text{length } xs \implies \text{sorted-sublist-wrt } R \text{ } xs \text{ } i \text{ } i \rangle$
by (*auto simp add: sorted-sublist-wrt-def sublist-single*)

lemma *sorted-sublist-refl*: $\langle i < \text{length } xs \implies \text{sorted-sublist } xs \text{ } i \text{ } i \rangle$
by (*auto simp add: sorted-sublist-def sorted-sublist-wrt-refl*)

lemma *sublist-map*: $\langle \text{sublist } (\text{map } f \text{ } xs) \text{ } i \text{ } j = \text{map } f \text{ } (\text{sublist } xs \text{ } i \text{ } j) \rangle$
apply (*auto simp add: sublist-def*)
by (*simp add: drop-map take-map*)

lemma *take-set*: $\langle j \leq \text{length } xs \implies x \in \text{set } (\text{take } j \text{ } xs) \equiv (\exists k. k < j \wedge xs!k = x) \rangle$
by (*rule eq-reflection*) (*auto simp add: take-set*)

lemma *drop-set*: $\langle j \leq \text{length } xs \implies x \in \text{set } (\text{drop } j \text{ } xs) \equiv (\exists k. j \leq k \wedge k < \text{length } xs \wedge xs!k = x) \rangle$
by (*smt Misc.in-set-drop-conv-nth*)

lemma *sublist-el*: $\langle i \leq j \implies j < \text{length } xs \implies x \in \text{set } (\text{sublist } xs \text{ } i \text{ } j) \equiv (\exists k. k < \text{Suc } j - i \wedge xs!(i+k) = x) \rangle$
by (*auto simp add: take-set sublist-def*)

lemma *sublist-el'*: $\langle i \leq j \implies j < \text{length } xs \implies x \in \text{set } (\text{sublist } xs \text{ } i \text{ } j) \equiv (\exists k. i \leq k \wedge k \leq j \wedge xs!k = x) \rangle$
apply (*subst sublist-el, assumption, assumption*)
by (*smt Groups.add-ac(2) le-add1 le-add-diff-inverse less-Suc-eq less-diff-conv nat-less-le order-refl*)

lemma *sublist-lt*: $\langle hi < lo \implies \text{sublist } xs \text{ } lo \text{ } hi = [] \rangle$
by (*auto simp add: sublist-def*)

lemma *nat-le-eq-or-lt*: $\langle a :: \text{nat} \rangle \leq b = (a = b \vee a < b)$
by *linarith*

lemma *sorted-sublist-wrt-le*: $\langle hi \leq lo \implies hi < \text{length } xs \implies \text{sorted-sublist-wrt } R \text{ } xs \text{ } lo \text{ } hi \rangle$
apply (*auto simp add: nat-le-eq-or-lt*)
unfolding *sorted-sublist-wrt-def*
subgoal **apply** (*rewrite sublist-single*) **by** *auto*

subgoal by (auto simp add: sublist-lt)
done

Elements in a sorted sublists are actually sorted

lemma sorted-sublist-wrt-nth-le:

assumes $\langle \text{sorted-sublist-wrt } R \text{ } xs \text{ } lo \text{ } hi \rangle$ and $\langle lo \leq hi \rangle$ and $\langle hi < \text{length } xs \rangle$ and
 $\langle lo \leq i \rangle$ and $\langle i < j \rangle$ and $\langle j \leq hi \rangle$
shows $\langle R (xs!i) (xs!j) \rangle$

proof –

have $A: \langle lo < \text{length } xs \rangle$ using *assms(2) assms(3)* by *linarith*

obtain i' where $I: \langle i = lo + i' \rangle$ using *assms(4) le-Suc-ex* by *auto*

obtain j' where $J: \langle j = lo + j' \rangle$ by (*meson assms(4) assms(5) dual-order.trans le-iff-add less-imp-le-nat*)

show *?thesis*

using *assms(1)* apply (*simp add: sorted-sublist-wrt-def I J*)

apply (*rewrite sublist-nth[symmetric, where k=i', where lo=lo, where hi=hi]*)

using *assms* apply *auto* subgoal using I by *linarith*

apply (*rewrite sublist-nth[symmetric, where k=j', where lo=lo, where hi=hi]*)

using *assms* apply *auto* subgoal using J by *linarith*

apply (*rule sorted-wrt-nth-less*)

apply *auto*

subgoal using $I J$ *nat-add-left-cancel-less* by *blast*

subgoal apply (*simp add: sublist-length*) using J by *linarith*

done

qed

We can make the assumption $i < j$ weaker if we have a reflexivie relation.

lemma sorted-sublist-wrt-nth-le':

assumes *ref*: $\langle \bigwedge x. R x x \rangle$

and $\langle \text{sorted-sublist-wrt } R \text{ } xs \text{ } lo \text{ } hi \rangle$ and $\langle lo \leq hi \rangle$ and $\langle hi < \text{length } xs \rangle$

and $\langle lo \leq i \rangle$ and $\langle i \leq j \rangle$ and $\langle j \leq hi \rangle$

shows $\langle R (xs!i) (xs!j) \rangle$

proof –

have $\langle i < j \vee i = j \rangle$ using $\langle i \leq j \rangle$ by *linarith*

then consider (a) $\langle i < j \rangle$ |

(b) $\langle i = j \rangle$ by *blast*

then show *?thesis*

proof *cases*

case a

then show *?thesis*

using *assms(2-5,7) sorted-sublist-wrt-nth-le* by *blast*

next

case b

then show *?thesis*

by (*simp add: ref*)

qed

qed

lemma sorted-sublist-le: $\langle hi \leq lo \implies hi < \text{length } xs \implies \text{sorted-sublist } xs \text{ } lo \text{ } hi \rangle$

by (*auto simp add: sorted-sublist-def sorted-sublist-wrt-le*)

lemma *sorted-sublist-map-le*: $\langle hi \leq lo \implies hi < \text{length } xs \implies \text{sorted-sublist-map } R \ h \ xs \ lo \ hi \rangle$
by (*auto simp add: sorted-sublist-wrt-le*)

lemma *sublist-cons*: $\langle lo < hi \implies hi < \text{length } xs \implies \text{sublist } xs \ lo \ hi = xs!lo \ \# \ \text{sublist } xs \ (\text{Suc } lo) \ hi \rangle$
by (*metis Cons-eq-appendI append-self-conv2 less-imp-le-nat less-or-eq-imp-le less-trans*
sublist-app sublist-single)

lemma *sorted-sublist-wrt-cons'*:
 $\langle \text{sorted-sublist-wrt } R \ xs \ (lo+1) \ hi \implies lo \leq hi \implies hi < \text{length } xs \implies (\forall j. lo < j \wedge j \leq hi \longrightarrow R \ (xs!lo) \ (xs!j)) \implies \text{sorted-sublist-wrt } R \ xs \ lo \ hi \rangle$
apply (*auto simp add: nat-le-eq-or-lt sorted-sublist-wrt-def*)
apply (*auto 5 4 simp add: sublist-cons sublist-el less-diff-conv add.commute[of - lo]*
dest: Suc-lessI sublist-single)
done

lemma *sorted-sublist-wrt-cons*:
assumes *trans*: $\langle (\bigwedge x \ y \ z. \llbracket R \ x \ y; R \ y \ z \rrbracket \implies R \ x \ z) \rangle$ **and**
 $\langle \text{sorted-sublist-wrt } R \ xs \ (lo+1) \ hi \rangle$ **and**
 $\langle lo \leq hi \rangle$ **and** $\langle hi < \text{length } xs \rangle$ **and** $\langle R \ (xs!lo) \ (xs!(lo+1)) \rangle$
shows $\langle \text{sorted-sublist-wrt } R \ xs \ lo \ hi \rangle$

proof –

show *?thesis*

apply (*rule sorted-sublist-wrt-cons'*) **using** *assms* **apply** *auto*
subgoal **premises** *assms'* **for** *j*

proof –

have *A*: $\langle j = lo+1 \vee j > lo+1 \rangle$ **using** *assms'(5)* **by** *linarith*

show *?thesis*

using *A* **proof**

assume *A*: $\langle j = lo+1 \rangle$ **show** *?thesis*

by (*simp add: A assms'*)

next

assume *A*: $\langle j > lo+1 \rangle$ **show** *?thesis*

apply (*rule trans*)

apply (*rule assms(5)*)

apply (*rule sorted-sublist-wrt-nth-le[OF assms(2), where i = lo+1, where j = j]*)

subgoal **using** *A assms'(6)* **by** *linarith*

subgoal **using** *assms'(3) less-imp-diff-less* **by** *blast*

subgoal **using** *assms'(5)* **by** *auto*

subgoal **using** *A* **by** *linarith*

subgoal **by** (*simp add: assms'(6)*)

done

qed

qed

done

qed

lemma *sorted-sublist-map-cons*:

$\langle (\bigwedge x \ y \ z. \llbracket R \ (h \ x) \ (h \ y); R \ (h \ y) \ (h \ z) \rrbracket \implies R \ (h \ x) \ (h \ z)) \implies$
 $\text{sorted-sublist-map } R \ h \ xs \ (lo+1) \ hi \implies lo \leq hi \implies hi < \text{length } xs \implies R \ (h \ (xs!lo)) \ (h \ (xs!(lo+1)))$
 $\implies \text{sorted-sublist-map } R \ h \ xs \ lo \ hi \rangle$
by (*blast intro: sorted-sublist-wrt-cons*)

lemma *sublist-snoc*: $\langle lo < hi \implies hi < \text{length } xs \implies \text{sublist } xs \ lo \ hi = \text{sublist } xs \ lo \ (hi-1) \ @ \ [xs!hi] \rangle$

apply (*simp add: sublist-def*)
proof –
assume $a1: lo < hi$
assume $hi < length\ xs$
then have $take\ lo\ xs @ take\ (Suc\ hi - lo)\ (drop\ lo\ xs) = (take\ lo\ xs @ take\ (hi - lo)\ (drop\ lo\ xs)) @ [xs ! hi]$
using $a1$ **by** (*metis (no-types) Suc-diff-le add-Suc-right hd-drop-conv-nth le-add-diff-inverse less-imp-le-nat take-add take-hd-drop*)
then show $take\ (Suc\ hi - lo)\ (drop\ lo\ xs) = take\ (hi - lo)\ (drop\ lo\ xs) @ [xs ! hi]$
by *simp*
qed

lemma *sorted-sublist-wrt-snoc'*:
 $\langle sorted-sublist-wrt\ R\ xs\ lo\ (hi-1) \implies lo \leq hi \implies hi < length\ xs \implies (\forall j. lo \leq j \wedge j < hi \longrightarrow R\ (xs!j)\ (xs!hi)) \implies sorted-sublist-wrt\ R\ xs\ lo\ hi \rangle$
apply (*simp add: sorted-sublist-wrt-def*)
apply (*auto simp add: nat-le-eq-or-lt*)
subgoal by (*simp add: sublist-single*)
by (*auto simp add: sublist-snoc sublist-el sorted-wrt-append add.commute[of lo] less-diff-conv simp: leI simp flip:nat-le-eq-or-lt*)

lemma *sorted-sublist-wrt-snoc*:
assumes *trans*: $\langle (\bigwedge x\ y\ z. \llbracket R\ x\ y; R\ y\ z \rrbracket \implies R\ x\ z) \rangle$ **and**
 $\langle sorted-sublist-wrt\ R\ xs\ lo\ (hi-1) \rangle$ **and**
 $\langle lo \leq hi \rangle$ **and** $\langle hi < length\ xs \rangle$ **and** $\langle (R\ (xs!(hi-1))\ (xs!hi)) \rangle$
shows $\langle sorted-sublist-wrt\ R\ xs\ lo\ hi \rangle$
proof –
show *?thesis*
apply (*rule sorted-sublist-wrt-snoc'*) **using** *assms* **apply** *auto*
subgoal premises *assms'* **for** j
proof –
have $A: \langle j=hi-1 \vee j < hi-1 \rangle$ **using** *assms'(6)* **by** *linarith*
show *?thesis*
using A **proof**
assume $A: \langle j=hi-1 \rangle$ **show** *?thesis*
by (*simp add: A assms'*)
next
assume $A: \langle j < hi-1 \rangle$ **show** *?thesis*
apply (*rule trans*)
apply (*rule sorted-sublist-wrt-nth-le[OF assms(2), where i=j, where j=hi-1]*)
prefer 6
apply (*rule assms(5)*)
apply *auto*
subgoal using A *assms'(5)* **by** *linarith*
subgoal using *assms'(3)* *less-imp-diff-less* **by** *blast*
subgoal using *assms'(5)* **by** *auto*
subgoal using A **by** *linarith*
done
qed
qed
done
qed

lemma *sublist-split*: $\langle lo \leq hi \implies lo < p \implies p < hi \implies hi < length\ xs \implies sublist\ xs\ lo\ p @ sublist\ xs$

$(p+1) \text{ hi} = \text{sublist } xs \text{ lo hi}$
by (*simp add: sublist-app*)

lemma *sublist-split-part*: $\langle lo \leq hi \implies lo < p \implies p < hi \implies hi < \text{length } xs \implies \text{sublist } xs \text{ lo } (p-1) @ xs!p \# \text{sublist } xs \text{ (p+1) hi} = \text{sublist } xs \text{ lo hi} \rangle$
by (*auto simp add: sublist-split[symmetric] sublist-snoc[where xs=xs,where lo=lo,where hi=p]*)

A property for partitions (we always assume that R is transitive).

lemma *isPartition-wrt-trans*:

$\langle (\bigwedge x y z. \llbracket R x y; R y z \rrbracket \implies R x z) \implies$
isPartition-wrt $R \text{ xs lo hi p} \implies$
 $(\forall i j. lo \leq i \wedge i < p \wedge p < j \wedge j \leq hi \longrightarrow R (xs!i) (xs!j)) \rangle$
by (*auto simp add: isPartition-wrt-def*)

lemma *isPartition-map-trans*:

$\langle (\bigwedge x y z. \llbracket R (h x) (h y); R (h y) (h z) \rrbracket \implies R (h x) (h z) \implies$
 $hi < \text{length } xs \implies$
isPartition-map $R h \text{ xs lo hi p} \implies$
 $(\forall i j. lo \leq i \wedge i < p \wedge p < j \wedge j \leq hi \longrightarrow R (h (xs!i)) (h (xs!j))) \rangle$
by (*auto simp add: isPartition-wrt-def*)

lemma *merge-sorted-wrt-partitions-between'*:

$\langle lo \leq hi \implies lo < p \implies p < hi \implies hi < \text{length } xs \implies$
isPartition-wrt $R \text{ xs lo hi p} \implies$
sorted-sublist-wrt $R \text{ xs lo } (p-1) \implies \text{sorted-sublist-wrt } R \text{ xs } (p+1) \text{ hi} \implies$
 $(\forall i j. lo \leq i \wedge i < p \wedge p < j \wedge j \leq hi \longrightarrow R (xs!i) (xs!j)) \implies$
sorted-sublist-wrt $R \text{ xs lo hi} \rangle$
apply (*auto simp add: isPartition-def isPartition-wrt-def sorted-sublist-def sorted-sublist-wrt-def sublist-map*)
apply (*simp add: sublist-split-part[symmetric]*)
apply (*auto simp add: List.sorted-wrt-append*)
subgoal by (*auto simp add: sublist-el*)
subgoal by (*auto simp add: sublist-el*)
subgoal by (*auto simp add: sublist-el'*)
done

lemma *merge-sorted-wrt-partitions-between*:

$\langle (\bigwedge x y z. \llbracket R x y; R y z \rrbracket \implies R x z) \implies$
isPartition-wrt $R \text{ xs lo hi p} \implies$
sorted-sublist-wrt $R \text{ xs lo } (p-1) \implies \text{sorted-sublist-wrt } R \text{ xs } (p+1) \text{ hi} \implies$
 $lo \leq hi \implies hi < \text{length } xs \implies lo < p \implies p < hi \implies hi < \text{length } xs \implies$
sorted-sublist-wrt $R \text{ xs lo hi} \rangle$
by (*simp add: merge-sorted-wrt-partitions-between' isPartition-wrt-trans*)

The main theorem to merge sorted lists

lemma *merge-sorted-wrt-partitions*:

$\langle \text{isPartition-wrt } R \text{ xs lo hi p} \implies$
sorted-sublist-wrt $R \text{ xs lo } (p - \text{Suc } 0) \implies \text{sorted-sublist-wrt } R \text{ xs } (\text{Suc } p) \text{ hi} \implies$
 $lo \leq hi \implies lo \leq p \implies p \leq hi \implies hi < \text{length } xs \implies$
 $(\forall i j. lo \leq i \wedge i < p \wedge p < j \wedge j \leq hi \longrightarrow R (xs!i) (xs!j)) \implies$
sorted-sublist-wrt $R \text{ xs lo hi} \rangle$
subgoal premises *assms*
proof –
have $C: \langle lo=p \wedge p=hi \vee lo=p \wedge p < hi \vee lo < p \wedge p=hi \vee lo < p \wedge p < hi \rangle$

```

using assms by linarith
show ?thesis
using C apply auto
subgoal — lo=p=hi
  apply (rule sorted-sublist-wrt-refl)
  using assms by auto
subgoal — lo=p<hi
  using assms by (simp add: isPartition-def isPartition-wrt-def sorted-sublist-wrt-cons')
subgoal — lo<p=hi
  using assms by (simp add: isPartition-def isPartition-wrt-def sorted-sublist-wrt-snoc')
subgoal — lo<p<hi
  using assms
  apply (rewrite merge-sorted-wrt-partitions-between'[where p=p])
  by auto
done
qed
done

```

theorem *merge-sorted-map-partitions:*

```

⟨(∧ x y z.  $\llbracket R (h x) (h y); R (h y) (h z) \rrbracket \implies R (h x) (h z) \implies$ 
  isPartition-map R h xs lo hi p  $\implies$ 
  sorted-sublist-map R h xs lo (p - Suc 0) \implies sorted-sublist-map R h xs (Suc p) hi \implies
  lo ≤ hi \implies lo ≤ p \implies p ≤ hi \implies hi < length xs \implies
  sorted-sublist-map R h xs lo hi⟩
apply (rule merge-sorted-wrt-partitions) apply auto
by (simp add: merge-sorted-wrt-partitions isPartition-map-trans)

```

lemma *partition-wrt-extend:*

```

⟨isPartition-wrt R xs lo' hi' p  $\implies$ 
  hi < length xs  $\implies$ 
  lo ≤ lo' \implies lo' ≤ hi \implies hi' ≤ hi \implies
  lo' ≤ p \implies p ≤ hi' \implies
  (∧ i. lo ≤ i \implies i < lo' \implies R (xs!i) (xs!p))  $\implies$ 
  (∧ j. hi' < j \implies j ≤ hi \implies R (xs!p) (xs!j))  $\implies$ 
  isPartition-wrt R xs lo hi p⟩
unfolding isPartition-wrt-def
apply (intro conjI)
subgoal
  by (force simp: not-le)
subgoal
  using leI by blast
done

```

lemma *partition-map-extend:*

```

⟨isPartition-map R h xs lo' hi' p  $\implies$ 
  hi < length xs  $\implies$ 
  lo ≤ lo' \implies lo' ≤ hi \implies hi' ≤ hi \implies
  lo' ≤ p \implies p ≤ hi' \implies
  (∧ i. lo ≤ i \implies i < lo' \implies R (h (xs!i)) (h (xs!p)))  $\implies$ 
  (∧ j. hi' < j \implies j ≤ hi \implies R (h (xs!p)) (h (xs!j)))  $\implies$ 
  isPartition-map R h xs lo hi p⟩
by (auto simp add: partition-wrt-extend)

```

lemma *isPartition-empty*:
 $\langle (\bigwedge j. \llbracket lo < j; j \leq hi \rrbracket \implies R (xs ! lo) (xs ! j)) \implies$
isPartition-wrt R xs lo hi lo
by (*auto simp add: isPartition-wrt-def*)

lemma *take-ext*:
 $\langle (\forall i < k. xs ! i = xs' ! i) \implies$
 $k < length\ xs \implies k < length\ xs' \implies$
 $take\ k\ xs' = take\ k\ xs \rangle$
by (*simp add: nth-take-lemma*)

lemma *drop-ext'*:
 $\langle (\forall i. i \geq k \wedge i < length\ xs \longrightarrow xs ! i = xs' ! i) \implies$
 $0 < k \implies xs \neq [] \implies$ — These corner cases will be dealt with in the next lemma
 $length\ xs' = length\ xs \implies$
 $drop\ k\ xs' = drop\ k\ xs \rangle$
apply (*rewrite in <drop - \sqsupset = -> List.rev-rev-ident[symmetric]*)
apply (*rewrite in <- = drop - \sqsupset > List.rev-rev-ident[symmetric]*)
apply (*rewrite in < \sqsupset = -> List.drop-rev*)
apply (*rewrite in <- = \sqsupset > List.drop-rev*)
apply *simp*
apply (*rule take-ext*)
by (*auto simp add: rev-nth*)

lemma *drop-ext*:
 $\langle (\forall i. i \geq k \wedge i < length\ xs \longrightarrow xs ! i = xs' ! i) \implies$
 $length\ xs' = length\ xs \implies$
 $drop\ k\ xs' = drop\ k\ xs \rangle$
apply (*cases xs*)
apply *auto*
apply (*cases k*)
subgoal by (*simp add: nth-equalityI*)
subgoal apply (*rule drop-ext'*) **by** *auto*
done

lemma *sublist-ext'*:
 $\langle (\forall i. lo \leq i \wedge i \leq hi \longrightarrow xs ! i = xs' ! i) \implies$
 $length\ xs' = length\ xs \implies$
 $lo \leq hi \implies Suc\ hi < length\ xs \implies$
 $sublist\ xs'\ lo\ hi = sublist\ xs\ lo\ hi \rangle$
apply (*simp add: sublist-def*)
apply (*rule take-ext*)
by *auto*

lemma *lt-Suc*: $\langle (a < b) = (Suc\ a = b \vee Suc\ a < b) \rangle$
by *auto*

lemma *sublist-until-end-eq-drop*: $\langle Suc\ hi = length\ xs \implies sublist\ xs\ lo\ hi = drop\ lo\ xs \rangle$
by (*simp add: sublist-def*)

lemma *sublist-ext*:

$\langle (\forall i. lo \leq i \wedge i \leq hi \longrightarrow xs!i = xs!i) \implies$
 $length\ xs' = length\ xs \implies$
 $lo \leq hi \implies hi < length\ xs \implies$
 $sublist\ xs'\ lo\ hi = sublist\ xs\ lo\ hi \rangle$
apply (auto simp add: lt-Suc[where a=hi])
subgoal by (auto simp add: sublist-until-end-eq-drop drop-ext)
subgoal by (auto simp add: sublist-ext')
done

lemma sorted-wrt-lower-sublist-still-sorted:

assumes $\langle sorted_sublist_wrt\ R\ xs\ lo\ (lo' - Suc\ 0) \rangle$ **and**
 $\langle lo \leq lo' \rangle$ **and** $\langle lo' < length\ xs \rangle$ **and**
 $\langle (\forall i. lo \leq i \wedge i < lo' \longrightarrow xs!i = xs!i) \rangle$ **and** $\langle length\ xs' = length\ xs \rangle$
shows $\langle sorted_sublist_wrt\ R\ xs'\ lo\ (lo' - Suc\ 0) \rangle$

proof –

have $l: \langle lo < lo' - 1 \vee lo \geq lo' - 1 \rangle$
by linarith
show ?thesis
using l **apply** auto
subgoal — $lo < lo' - 1$
apply (auto simp add: sorted-sublist-wrt-def)
apply (rewrite sublist-ext[where xs=xs])
using *assms* **by** (auto simp add: sorted-sublist-wrt-def)
subgoal — $lo \geq lo' - 1$
using *assms* **by** (auto simp add: sorted-sublist-wrt-le)
done

qed

lemma sorted-map-lower-sublist-still-sorted:

assumes $\langle sorted_sublist_map\ R\ h\ xs\ lo\ (lo' - Suc\ 0) \rangle$ **and**
 $\langle lo \leq lo' \rangle$ **and** $\langle lo' < length\ xs \rangle$ **and**
 $\langle (\forall i. lo \leq i \wedge i < lo' \longrightarrow xs!i = xs!i) \rangle$ **and** $\langle length\ xs' = length\ xs \rangle$
shows $\langle sorted_sublist_map\ R\ h\ xs'\ lo\ (lo' - Suc\ 0) \rangle$
using *assms* **by** (rule sorted-wrt-lower-sublist-still-sorted)

lemma sorted-wrt-upper-sublist-still-sorted:

assumes $\langle sorted_sublist_wrt\ R\ xs\ (hi'+1)\ hi \rangle$ **and**
 $\langle lo \leq lo' \rangle$ **and** $\langle hi < length\ xs \rangle$ **and**
 $\langle \forall j. hi' < j \wedge j \leq hi \longrightarrow xs!j = xs!j \rangle$ **and** $\langle length\ xs' = length\ xs \rangle$
shows $\langle sorted_sublist_wrt\ R\ xs'\ (hi'+1)\ hi \rangle$

proof –

have $l: \langle hi' + 1 < hi \vee hi' + 1 \geq hi \rangle$
by linarith
show ?thesis
using l **apply** auto
subgoal — $hi' + 1 < hi$
apply (auto simp add: sorted-sublist-wrt-def)
apply (rewrite sublist-ext[where xs=xs])
using *assms* **by** (auto simp add: sorted-sublist-wrt-def)
subgoal — $hi \leq hi' + 1$
using *assms* **by** (auto simp add: sorted-sublist-wrt-le)
done

qed

lemma sorted-map-upper-sublist-still-sorted:

assumes $\langle \text{sorted-sublist-map } R \ h \ xs \ (hi'+1) \ hi \rangle$ **and**
 $\langle lo \leq lo' \rangle$ **and** $\langle hi < \text{length } xs \rangle$ **and**
 $\langle \forall j. hi' < j \wedge j \leq hi \longrightarrow xs!j = xs!j \rangle$ **and** $\langle \text{length } xs' = \text{length } xs \rangle$
shows $\langle \text{sorted-sublist-map } R \ h \ xs' \ (hi'+1) \ hi \rangle$
using *assms* **by** (rule *sorted-wrt-upper-sublist-still-sorted*)

The specification of the partition function

definition *partition-spec* :: $\langle ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \text{ list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ list} \Rightarrow \text{nat} \Rightarrow \text{bool} \rangle$ **where**

$\langle \text{partition-spec } R \ h \ xs \ lo \ hi \ xs' \ p \equiv$
 $\text{mset } xs' = \text{mset } xs \wedge \text{--- The list is a permutation}$
 $\text{isPartition-map } R \ h \ xs' \ lo \ hi \ p \wedge \text{--- We have a valid partition on the resulting list}$
 $lo \leq p \wedge p \leq hi \wedge \text{--- The partition index is in bounds}$
 $(\forall i. i < lo \longrightarrow xs!i = xs!i) \wedge (\forall i. hi < i \wedge i < \text{length } xs' \longrightarrow xs!i = xs!i) \rangle$ **--- Everything else is unchanged.**

lemma *in-set-take-conv-nth*:

$\langle x \in \text{set } (\text{take } n \ xs) \longleftrightarrow (\exists m < \min n \ (\text{length } xs). \ xs!m = x) \rangle$
by (*metis in-set-conv-nth length-take min.commute min.strict-boundedE nth-take*)

lemma *mset-drop-upto*: $\langle \text{mset } (\text{drop } a \ N) = \{ \#N!i. i \in \# \text{mset-set } \{a..<\text{length } N\} \# \} \rangle$

proof (*induction N arbitrary: a*)

case *Nil*

then show *?case* **by** *simp*

next

case (*Cons c N*)

have *upt*: $\langle \{0..<\text{Suc } (\text{length } N)\} = \text{insert } 0 \ \{1..<\text{Suc } (\text{length } N)\} \rangle$

by *auto*

then have *H*: $\langle \text{mset-set } \{0..<\text{Suc } (\text{length } N)\} = \text{add-mset } 0 \ (\text{mset-set } \{1..<\text{Suc } (\text{length } N)\}) \rangle$

unfolding *upt* **by** *auto*

have *mset-case-Suc*: $\langle \{ \# \text{case } x \text{ of } 0 \Rightarrow c \mid \text{Suc } x \Rightarrow N!x . x \in \# \text{mset-set } \{ \text{Suc } a..<\text{Suc } b \} \# \} = \{ \#N! (x-1) . x \in \# \text{mset-set } \{ \text{Suc } a..<\text{Suc } b \} \# \} \rangle$ **for** *a b*

by (*rule image-mset-cong*) (*auto split: nat.splits*)

have *Suc-Suc*: $\langle \{ \text{Suc } a..<\text{Suc } b \} = \text{Suc } \{ a..<b \} \rangle$ **for** *a b*

by *auto*

then have *mset-set-Suc-Suc*: $\langle \text{mset-set } \{ \text{Suc } a..<\text{Suc } b \} = \{ \# \text{Suc } n. n \in \# \text{mset-set } \{ a..<b \} \# \} \rangle$ **for** *a b*

unfolding *Suc-Suc* **by** (*subst image-mset-mset-set[symmetric]*) *auto*

have ***: $\langle \{ \#N! (x-\text{Suc } 0) . x \in \# \text{mset-set } \{ \text{Suc } a..<\text{Suc } b \} \# \} = \{ \#N! x . x \in \# \text{mset-set } \{ a..<b \} \# \} \rangle$

for *a b*

by (*auto simp add: mset-set-Suc-Suc multiset.map-comp comp-def*)

show *?case*

apply (*cases a*)

using *Cons[of 0] Cons* **by** (*auto simp: nth-Cons drop-Cons H mset-case-Suc **)

qed

lemma *mathias*:

assumes

Perm: $\langle \text{mset } xs' = \text{mset } xs \rangle$

and *I*: $\langle lo \leq i \rangle \langle i \leq hi \rangle \langle xs!i = x \rangle$

and *Bounds*: $\langle hi < \text{length } xs \rangle$

and *Fix*: $\langle \bigwedge i. i < lo \implies xs!i = xs!i \rangle \langle \bigwedge j. \llbracket hi < j; j < \text{length } xs \rrbracket \implies xs!j = xs!j \rangle$

shows $\langle \exists j. lo \leq j \wedge j \leq hi \wedge xs!j = x \rangle$

proof **–**

define *xs1 xs2 xs3 xs1' xs2' xs3'* **where**

```

  ⟨xs1 = take lo xs⟩ and
  ⟨xs2 = take (Suc hi - lo) (drop lo xs)⟩ and
  ⟨xs3 = drop (Suc hi) xs⟩ and
  ⟨xs1' = take lo xs'⟩ and
  ⟨xs2' = take (Suc hi - lo) (drop lo xs')⟩ and
  ⟨xs3' = drop (Suc hi) xs'⟩
have [simp]: ⟨length xs' = length xs⟩
  using Perm by (auto dest: mset-eq-length)
have [simp]: ⟨mset xs1 = mset xs1'⟩
  using Fix(1) unfolding xs1-def xs1'-def
  by (metis Perm le-cases mset-eq-length nth-take-lemma take-all)
have [simp]: ⟨mset xs3 = mset xs3'⟩
  using Fix(2) unfolding xs3-def xs3'-def mset-drop-upto
  by (auto intro: image-mset-cong)
have ⟨xs = xs1 @ xs2 @ xs3⟩ ⟨xs' = xs1' @ xs2' @ xs3'⟩
  using I unfolding xs1-def xs2-def xs3-def xs1'-def xs2'-def xs3'-def
  by (metis append.assoc append-take-drop-id le-SucI le-add-diff-inverse order-trans take-add)+
moreover have ⟨xs ! i = xs2 ! (i - lo)⟩ ⟨i ≥ length xs1⟩
  using I Bounds unfolding xs2-def xs1-def by (auto simp: nth-take min-def)
moreover have ⟨x ∈ set xs2'⟩
  using I Bounds unfolding xs2'-def
  by (auto simp: in-set-take-conv-nth
    intro!: exI[of - ⟨i - lo⟩])
ultimately have ⟨x ∈ set xs2⟩
  using Perm I by (auto dest: mset-eq-setD)
then obtain j where ⟨xs ! (lo + j) = x⟩ ⟨j ≤ hi - lo⟩
  unfolding in-set-conv-nth xs2-def
  by auto
then show ?thesis
  using Bounds I
  by (auto intro: exI[of - ⟨lo+j⟩])
qed

```

If we fix the left and right rest of two permutated lists, then the sublists are also permutations.

But we only need that the sets are equal.

lemma *mset-sublist-incl*:

```

  assumes Perm: ⟨mset xs' = mset xs⟩
  and Fix: ⟨ $\bigwedge i. i < lo \implies xs ! i = xs' ! i$ ⟩ ⟨ $\bigwedge j. \llbracket hi < j; j < length xs \rrbracket \implies xs' ! j = xs ! j$ ⟩
  and bounds: ⟨lo ≤ hi⟩ ⟨hi < length xs⟩
  shows ⟨set (sublist xs' lo hi) ⊆ set (sublist xs lo hi)⟩

```

proof

```

  fix x
  assume ⟨x ∈ set (sublist xs' lo hi)⟩
  then have ⟨ $\exists i. lo \leq i \wedge i \leq hi \wedge xs' ! i = x$ ⟩
    by (metis assms(1) bounds(1) bounds(2) size-mset sublist-el')
  then obtain i where I: ⟨lo ≤ i⟩ ⟨i ≤ hi⟩ ⟨xs' ! i = x⟩ by blast
  have ⟨ $\exists j. lo \leq j \wedge j \leq hi \wedge xs ! j = x$ ⟩
    using Perm I bounds(2) Fix by (rule mathias, auto)
  then show ⟨x ∈ set (sublist xs lo hi)⟩
    by (simp add: bounds(1) bounds(2) sublist-el')

```

qed

lemma *mset-sublist-eq*:

```

  assumes ⟨mset xs' = mset xs⟩

```

```

and  $\langle \bigwedge i. i < lo \implies xs!i = xs!i \rangle$ 
and  $\langle \bigwedge j. \llbracket hi < j; j < length\ xs \rrbracket \implies xs!j = xs!j \rangle$ 
and bounds:  $\langle lo \leq hi \rangle \langle hi < length\ xs \rangle$ 
shows  $\langle set\ (sublist\ xs'\ lo\ hi) = set\ (sublist\ xs\ lo\ hi) \rangle$ 
proof
show  $\langle set\ (sublist\ xs'\ lo\ hi) \subseteq set\ (sublist\ xs\ lo\ hi) \rangle$ 
apply (rule mset-sublist-incl)
using assms by auto
show  $\langle set\ (sublist\ xs\ lo\ hi) \subseteq set\ (sublist\ xs'\ lo\ hi) \rangle$ 
by (rule mset-sublist-incl) (metis assms size-mset)+
qed

```

Our abstract recursive quicksort procedure. We abstract over a partition procedure.

```

definition quicksort ::  $\langle ('b \implies 'b \implies bool) \implies ('a \implies 'b) \implies nat \times nat \times 'a\ list \implies 'a\ list\ nres \rangle$  where
 $\langle quicksort\ R\ h = (\lambda(lo,hi,xs0). do\ \{$ 
  RECT  $(\lambda f\ (lo,hi,xs). do\ \{$ 
    ASSERT  $(lo \leq hi \wedge hi < length\ xs \wedge mset\ xs = mset\ xs0)$ ; — Premise for a partition function
     $(xs, p) \leftarrow SPEC(uncurry\ (partition-spec\ R\ h\ xs\ lo\ hi))$ ; — Abstract partition function
    ASSERT  $(mset\ xs = mset\ xs0)$ ;
     $xs \leftarrow (if\ p-1 \leq lo\ then\ RETURN\ xs\ else\ f\ (lo, p-1, xs))$ ;
    ASSERT  $(mset\ xs = mset\ xs0)$ ;
     $if\ hi \leq p+1\ then\ RETURN\ xs\ else\ f\ (p+1, hi, xs)$ 
   $\})\ (lo,hi,xs0)$ 
 $\}) \rangle$ 

```

As premise for quicksor, we only need that the indices are ok.

```

definition quicksort-pre ::  $\langle ('b \implies 'b \implies bool) \implies ('a \implies 'b) \implies 'a\ list \implies nat \implies nat \implies 'a\ list \implies bool \rangle$ 
where
 $\langle quicksort-pre\ R\ h\ xs0\ lo\ hi\ xs \equiv lo \leq hi \wedge hi < length\ xs \wedge mset\ xs = mset\ xs0 \rangle$ 

```

```

definition quicksort-post ::  $\langle ('b \implies 'b \implies bool) \implies ('a \implies 'b) \implies nat \implies nat \implies 'a\ list \implies 'a\ list \implies bool \rangle$ 
where
 $\langle quicksort-post\ R\ h\ lo\ hi\ xs\ xs' \equiv$ 
   $mset\ xs' = mset\ xs \wedge$ 
   $sorted-sublist-map\ R\ h\ xs'\ lo\ hi \wedge$ 
   $(\forall i. i < lo \longrightarrow xs!i = xs!i) \wedge$ 
   $(\forall j. hi < j \wedge j < length\ xs \longrightarrow xs!j = xs!j) \rangle$ 

```

Convert Pure to HOL

```

lemma quicksort-postI:
 $\langle \llbracket mset\ xs' = mset\ xs; sorted-sublist-map\ R\ h\ xs'\ lo\ hi; (\bigwedge i. \llbracket i < lo \rrbracket \implies xs!i = xs!i); (\bigwedge j. \llbracket hi < j; j < length\ xs \rrbracket \implies xs!j = xs!j) \rrbracket \implies quicksort-post\ R\ h\ lo\ hi\ xs\ xs' \rangle$ 
by (auto simp add: quicksort-post-def)

```

The first case for the correctness proof of (abstract) quicksort: We assume that we called the partition function, and we have $p - (1::'a) \leq lo$ and $hi \leq p + (1::'a)$.

```

lemma quicksort-correct-case1:
assumes trans:  $\langle \bigwedge x\ y\ z. \llbracket R\ (h\ x)\ (h\ y); R\ (h\ y)\ (h\ z) \rrbracket \implies R\ (h\ x)\ (h\ z) \rangle$  and lin:  $\langle \bigwedge x\ y. x \neq y \implies R\ (h\ x)\ (h\ y) \vee R\ (h\ y)\ (h\ x) \rangle$ 
and pre:  $\langle quicksort-pre\ R\ h\ xs0\ lo\ hi\ xs \rangle$ 
and part:  $\langle partition-spec\ R\ h\ xs\ lo\ hi\ xs'\ p \rangle$ 
and ifs:  $\langle p-1 \leq lo \rangle \langle hi \leq p+1 \rangle$ 
shows  $\langle quicksort-post\ R\ h\ lo\ hi\ xs\ xs' \rangle$ 
proof —

```

First boilerplate code step: 'unfold' the HOL definitions in the assumptions and convert them to Pure

```

have pre: ⟨lo ≤ hi⟩ ⟨hi < length xs⟩
  using pre by (auto simp add: quicksort-pre-def)

have part: ⟨mset xs' = mset xs⟩ True
  ⟨isPartition-map R h xs' lo hi p⟩ ⟨lo ≤ p⟩ ⟨p ≤ hi⟩
  ⟨∧ i. i < lo ⇒ xs!i = xs!i⟩ ⟨∧ i. [[hi < i; i < length xs]] ⇒ xs!i = xs!i⟩
  using part by (auto simp add: partition-spec-def)

have sorted-lower: ⟨sorted-sublist-map R h xs' lo (p - Suc 0)⟩
proof -
  show ?thesis
  apply (rule sorted-sublist-wrt-le)
  subgoal using ifs(1) by auto
  subgoal using ifs(1) mset-eq-length part(1) pre(1) pre(2) by fastforce
  done
qed

have sorted-upper: ⟨sorted-sublist-map R h xs' (Suc p) hi⟩
proof -
  show ?thesis
  apply (rule sorted-sublist-wrt-le)
  subgoal using ifs(2) by auto
  subgoal using ifs(1) mset-eq-length part(1) pre(1) pre(2) by fastforce
  done
qed

have sorted-middle: ⟨sorted-sublist-map R h xs' lo hi⟩
proof -
  show ?thesis
  apply (rule merge-sorted-map-partitions[where p=p])
  subgoal by (rule trans)
  subgoal by (rule part)
  subgoal by (rule sorted-lower)
  subgoal by (rule sorted-upper)
  subgoal using pre(1) by auto
  subgoal by (simp add: part(4))
  subgoal by (simp add: part(5))
  subgoal by (metis part(1) pre(2) size-mset)
  done
qed

show ?thesis
proof (intro quicksort-postI)
  show ⟨mset xs' = mset xs⟩
  by (simp add: part(1))
next
  show ⟨sorted-sublist-map R h xs' lo hi⟩
  by (rule sorted-middle)
next
  show ⟨∧ i. i < lo ⇒ xs' ! i = xs ! i⟩
  using part(6) by blast
next

```

```

  show  $\langle \bigwedge j. \llbracket hi < j; j < length\ xs \rrbracket \implies xs' ! j = xs ! j \rangle$ 
    by (metis part(1) part(7) size-mset)
qed
qed

```

In the second case, we have to show that the precondition still holds for $(p+1, hi, x')$ after the partition.

```

lemma quicksort-correct-case2:
  assumes
    pre:  $\langle quicksort-pre\ R\ h\ xs0\ lo\ hi\ xs \rangle$ 
    and part:  $\langle partition-spec\ R\ h\ xs\ lo\ hi\ xs'\ p \rangle$ 
    and ifs:  $\langle \neg hi \leq p + 1 \rangle$ 
  shows  $\langle quicksort-pre\ R\ h\ xs0\ (Suc\ p)\ hi\ xs' \rangle$ 
proof -

```

First boilerplate code step: 'unfold' the HOL definitions in the assumptions and convert them to Pure

```

  have pre:  $\langle lo \leq hi \rangle \langle hi < length\ xs \rangle \langle mset\ xs = mset\ xs0 \rangle$ 
    using pre by (auto simp add: quicksort-pre-def)
  have part:  $\langle mset\ xs' = mset\ xs \rangle\ True$ 
     $\langle isPartition-map\ R\ h\ xs'\ lo\ hi\ p \rangle \langle lo \leq p \rangle \langle p \leq hi \rangle$ 
     $\langle \bigwedge i. i < lo \implies xs' ! i = xs ! i \rangle \langle \bigwedge i. \llbracket hi < i; i < length\ xs \rrbracket \implies xs' ! i = xs ! i \rangle$ 
    using part by (auto simp add: partition-spec-def)
  show ?thesis
    unfolding quicksort-pre-def
  proof (intro conjI)
    show  $\langle Suc\ p \leq hi \rangle$ 
      using ifs by linarith
    show  $\langle hi < length\ xs' \rangle$ 
      by (metis part(1) pre(2) size-mset)
    show  $\langle mset\ xs' = mset\ xs0 \rangle$ 
      using pre(3) part(1) by (auto dest: mset-eq-setD)
  qed
qed

```

```

lemma quicksort-post-set:
  assumes  $\langle quicksort-post\ R\ h\ lo\ hi\ xs\ xs' \rangle$ 
    and bounds:  $\langle lo \leq hi \rangle \langle hi < length\ xs \rangle$ 
  shows  $\langle set\ (sublist\ xs'\ lo\ hi) = set\ (sublist\ xs\ lo\ hi) \rangle$ 
proof -
  have  $\langle mset\ xs' = mset\ xs \rangle \langle \bigwedge i. i < lo \implies xs' ! i = xs ! i \rangle \langle \bigwedge j. \llbracket hi < j; j < length\ xs \rrbracket \implies xs' ! j = xs ! j \rangle$ 
    using assms by (auto simp add: quicksort-post-def)
  then show ?thesis
    using bounds by (rule mset-sublist-eq, auto)
qed

```

In the third case, we have run quicksort recursively on $(p+1, hi, xs')$ after the partition, with $hi \leq p+1$ and $p-1 \leq lo$.

```

lemma quicksort-correct-case3:
  assumes trans:  $\langle \bigwedge x\ y\ z. \llbracket R\ (h\ x)\ (h\ y); R\ (h\ y)\ (h\ z) \rrbracket \implies R\ (h\ x)\ (h\ z) \rangle$  and lin:  $\langle \bigwedge x\ y. x \neq y \implies R\ (h\ x)\ (h\ y) \vee R\ (h\ y)\ (h\ x) \rangle$ 
    and pre:  $\langle quicksort-pre\ R\ h\ xs0\ lo\ hi\ xs \rangle$ 
    and part:  $\langle partition-spec\ R\ h\ xs\ lo\ hi\ xs'\ p \rangle$ 

```

```

  and ifs: ⟨p - Suc 0 ≤ lo⟩ ⟨¬ hi ≤ Suc p⟩
  and IH1': ⟨quicksort-post R h (Suc p) hi xs' xs''⟩
  shows ⟨quicksort-post R h lo hi xs xs''⟩
proof -

```

First boilerplate code step: 'unfold' the HOL definitions in the assumptions and convert them to Pure

```

have pre: ⟨lo ≤ hi⟩ ⟨hi < length xs⟩ ⟨mset xs = mset xs0⟩
  using pre by (auto simp add: quicksort-pre-def)
have part: ⟨mset xs' = mset xs⟩ True
  ⟨isPartition-map R h xs' lo hi p⟩ ⟨lo ≤ p⟩ ⟨p ≤ hi⟩
  ⟨∧ i. i < lo ⇒ xs!i = xs'i⟩ ⟨∧ i. [hi < i; i < length xs] ⇒ xs!i = xs'i⟩
  using part by (auto simp add: partition-spec-def)
have IH1: ⟨mset xs'' = mset xs'⟩ ⟨sorted-sublist-map R h xs'' (Suc p) hi⟩
  ⟨∧ i. i < Suc p ⇒ xs''!i = xs'!i⟩ ⟨∧ j. [hi < j; j < length xs] ⇒ xs''!j = xs'!j⟩
  using IH1' by (auto simp add: quicksort-post-def)
note IH1-perm = quicksort-post-set[OF IH1]

have still-partition: ⟨isPartition-map R h xs'' lo hi p⟩
proof (intro isPartition-wrtI)
  fix i assume ⟨lo ≤ i⟩ ⟨i < p⟩
  show ⟨R (h (xs''!i)) (h (xs''!p))⟩

```

This holds because this part hasn't changed

```

  using IH1(3) ⟨i < p⟩ ⟨lo ≤ i⟩ isPartition-wrt-def part(3) by fastforce
next
  fix j assume ⟨p < j⟩ ⟨j ≤ hi⟩

```

Obtain the position *posJ* where *xs''!j* was stored in *xs'*.

```

  have ⟨xs''!j ∈ set (sublist xs'' (Suc p) hi)⟩
  by (metis IH1(1) Suc-leI ⟨j ≤ hi⟩ ⟨p < j⟩ less-le-trans mset-eq-length part(1) pre(2) sublist-el')
  then have ⟨xs''!j ∈ set (sublist xs' (Suc p) hi)⟩
  by (metis IH1-perm ifs(2) nat-le-linear part(1) pre(2) size-mset)
  then have ⟨∃ posJ. Suc p ≤ posJ ∧ posJ ≤ hi ∧ xs''!j = xs!posJ⟩
  by (metis Suc-leI ⟨j ≤ hi⟩ ⟨p < j⟩ less-le-trans part(1) pre(2) size-mset sublist-el')
  then obtain posJ :: nat where PosJ: ⟨Suc p ≤ posJ⟩ ⟨posJ ≤ hi⟩ ⟨xs''!j = xs!posJ⟩ by blast

  then show ⟨R (h (xs''!p)) (h (xs''!j))⟩
  by (metis IH1(3) Suc-le-lessD isPartition-wrt-def lessI part(3))
qed

```

```

have sorted-lower: ⟨sorted-sublist-map R h xs'' lo (p - Suc 0)⟩
proof -
  show ?thesis
  apply (rule sorted-sublist-wrt-le)
  subgoal by (simp add: ifs(1))
  subgoal using IH1(1) mset-eq-length part(1) part(5) pre(2) by fastforce
  done
qed

```

```

note sorted-upper = IH1(2)

```

```

have sorted-middle: ⟨sorted-sublist-map R h xs'' lo hi⟩
proof -
  show ?thesis

```

```

apply (rule merge-sorted-map-partitions[where p=p])
subgoal by (rule trans)
subgoal by (rule still-partition)
subgoal by (rule sorted-lower)
subgoal by (rule sorted-upper)
subgoal using pre(1) by auto
subgoal by (simp add: part(4))
subgoal by (simp add: part(5))
subgoal by (metis IH1(1) part(1) pre(2) size-mset)
done
qed

show ?thesis
proof (intro quicksort-postI)
  show ⟨mset xs'' = mset xs⟩
    using part(1) IH1(1) by auto — I was faster than sledgehammer :-
next
  show ⟨sorted-sublist-map R h xs'' lo hi⟩
    by (rule sorted-middle)
next
  show ⟨ $\bigwedge i. i < lo \implies xs'' ! i = xs ! i$ ⟩
    using IH1(3) le-SucI part(4) part(6) by auto
next show ⟨ $\bigwedge j. hi < j \implies j < \text{length } xs \implies xs'' ! j = xs ! j$ ⟩
    by (metis IH1(4) part(1) part(7) size-mset)
qed
qed

```

In the 4th case, we have to show that the premise holds for $(lo, p - (1::'b), xs')$, in case $\neg p - (1::'a) \leq lo$

Analogous to case 2.

lemma quicksort-correct-case4:

```

assumes
  pre: ⟨quicksort-pre R h xs0 lo hi xs⟩
  and part: ⟨partition-spec R h xs lo hi xs' p⟩
  and ifs: ⟨ $\neg p - \text{Suc } 0 \leq lo$ ⟩
shows ⟨quicksort-pre R h xs0 lo (p - Suc 0) xs'⟩
proof —

```

First boilerplate code step: 'unfold' the HOL definitions in the assumptions and convert them to Pure

```

have pre: ⟨lo ≤ hi⟩ ⟨hi < length xs⟩ ⟨mset xs0 = mset xs⟩
  using pre by (auto simp add: quicksort-pre-def)
have part: ⟨mset xs' = mset xs⟩ True
  ⟨isPartition-map R h xs' lo hi p⟩ ⟨lo ≤ p⟩ ⟨p ≤ hi⟩
  ⟨ $\bigwedge i. i < lo \implies xs' ! i = xs ! i$ ⟩ ⟨ $\bigwedge i. \llbracket hi < i; i < \text{length } xs \rrbracket \implies xs' ! i = xs ! i$ ⟩
  using part by (auto simp add: partition-spec-def)

show ?thesis
unfolding quicksort-pre-def
proof (intro conjI)
  show ⟨lo ≤ p - Suc 0⟩
    using ifs by linarith
  show ⟨p - Suc 0 < length xs'⟩

```

```

    using mset-eq-length part(1) part(5) pre(2) by fastforce
  show ⟨mset xs' = mset xs0⟩
    using pre(3) part(1) by (auto dest: mset-eq-setD)
qed
qed

```

In the 5th case, we have run quicksort recursively on (lo, p-1, xs').

lemma *quicksort-correct-case5*:

```

  assumes trans: ⟨ $\bigwedge x y z. \llbracket R(h x) (h y); R(h y) (h z) \rrbracket \implies R(h x) (h z)$ ⟩ and lin: ⟨ $\bigwedge x y. x \neq y \implies R(h x) (h y) \vee R(h y) (h x)$ ⟩
  and pre: ⟨quicksort-pre R h xs0 lo hi xs⟩
  and part: ⟨partition-spec R h xs lo hi xs' p⟩
  and ifs: ⟨ $\neg p - \text{Suc } 0 \leq \text{lo}$ ⟩ ⟨hi ≤ Suc p⟩
  and IH1': ⟨quicksort-post R h lo (p - Suc 0) xs' xs''⟩
  shows ⟨quicksort-post R h lo hi xs xs''⟩

```

proof –

First boilerplate code step: 'unfold' the HOL definitions in the assumptions and convert them to Pure

```

  have pre: ⟨lo ≤ hi⟩ ⟨hi < length xs⟩
    using pre by (auto simp add: quicksort-pre-def)
  have part: ⟨mset xs' = mset xs⟩ True
    ⟨isPartition-map R h xs' lo hi p⟩ ⟨lo ≤ p⟩ ⟨p ≤ hi⟩
    ⟨ $\bigwedge i. i < \text{lo} \implies xs' ! i = xs ! i$ ⟩ ⟨ $\bigwedge i. \llbracket hi < i; i < \text{length } xs \rrbracket \implies xs' ! i = xs ! i$ ⟩
    using part by (auto simp add: partition-spec-def)
  have IH1: ⟨mset xs'' = mset xs'⟩ ⟨sorted-sublist-map R h xs'' lo (p - Suc 0)⟩
    ⟨ $\bigwedge i. i < \text{lo} \implies xs'' ! i = xs' ! i$ ⟩ ⟨ $\bigwedge j. \llbracket p - \text{Suc } 0 < j; j < \text{length } xs \rrbracket \implies xs'' ! j = xs' ! j$ ⟩
    using IH1' by (auto simp add: quicksort-post-def)
  note IH1-perm = quicksort-post-set[OF IH1]

```

```

  have still-partition: ⟨isPartition-map R h xs'' lo hi p⟩
  proof(intro isPartition-wrtI)
    fix i assume ⟨lo ≤ i⟩ ⟨i < p⟩

```

Obtain the position *posI* where *xs'' ! i* was stored in *xs'*.

```

  have ⟨xs'' ! i ∈ set (sublist xs'' lo (p - Suc 0))⟩
    by (metis (no-types, lifting) IH1(1) Suc-leI Suc-pred ⟨i < p⟩ ⟨lo ≤ i⟩ le-less-trans less-imp-diff-less
    mset-eq-length not-le not-less-zero part(1) part(5) pre(2) sublist-el')
  then have ⟨xs'' ! i ∈ set (sublist xs' lo (p - Suc 0))⟩
    by (metis IH1-perm ifs(1) le-less-trans less-imp-diff-less mset-eq-length nat-le-linear part(1)
    part(5) pre(2))
  then have ⟨ $\exists \text{ posI}. \text{lo} \leq \text{posI} \wedge \text{posI} \leq p - \text{Suc } 0 \wedge xs'' ! i = xs' ! \text{posI}$ ⟩
  proof – — sledgehammer
    have p - Suc 0 < length xs
      by (meson diff-le-self le-less-trans part(5) pre(2))
    then show ?thesis
      by (metis (no-types) ⟨xs'' ! i ∈ set (sublist xs' lo (p - Suc 0))⟩ ifs(1) mset-eq-length nat-le-linear
      part(1) sublist-el')
  qed
  then obtain posI :: nat where PosI: ⟨lo ≤ posI⟩ ⟨posI ≤ p - Suc 0⟩ ⟨xs'' ! i = xs' ! posI⟩ by blast
  then show ⟨R(h (xs'' ! i)) (h (xs' ! posI))⟩
    by (metis (no-types, lifting) IH1(4) ⟨i < p⟩ diff-Suc-less isPartition-wrt-def le-less-trans
    mset-eq-length not-le not-less-eq part(1) part(3) part(5) pre(2) zero-less-Suc)
  next

```



```

fix j assume ⟨p < j⟩ ⟨j ≤ hi⟩
then show ⟨R (h (xs'' ! p)) (h (xs'' ! j))⟩

```

This holds because this part hasn't changed

```

by (smt IH1(4) add-diff-cancel-left' add-diff-inverse-nat diff-Suc-eq-diff-pred diff-le-self ifs(1) isPartition-wrt-def le-less-Suc-eq less-le-trans mset-eq-length nat-less-le part(1) part(3) part(4) plus-1-eq-Suc pre(2))
qed

```

```

note sorted-lower = IH1(2)

```

```

have sorted-upper: ⟨sorted-sublist-map R h xs'' (Suc p) hi⟩

```

```

proof –
  show ?thesis
    apply (rule sorted-sublist-wrt-le)
    subgoal by (simp add: ifs(2))
    subgoal using IH1(1) mset-eq-length part(1) part(5) pre(2) by fastforce
    done
qed

```

```

have sorted-middle: ⟨sorted-sublist-map R h xs'' lo hi⟩

```

```

proof –
  show ?thesis
    apply (rule merge-sorted-map-partitions[where p=p])
    subgoal by (rule trans)
    subgoal by (rule still-partition)
    subgoal by (rule sorted-lower)
    subgoal by (rule sorted-upper)
    subgoal using pre(1) by auto
    subgoal by (simp add: part(4))
    subgoal by (simp add: part(5))
    subgoal by (metis IH1(1) part(1) pre(2) size-mset)
    done
qed

```

```

show ?thesis

```

```

proof (intro quicksort-postI)

```

```

  show ⟨mset xs'' = mset xs⟩
  by (simp add: IH1(1) part(1))

```

```

next

```

```

  show ⟨sorted-sublist-map R h xs'' lo hi⟩
  by (rule sorted-middle)

```

```

next

```

```

  show ⟨ $\bigwedge i. i < lo \implies xs'' ! i = xs ! i$ ⟩
  by (simp add: IH1(3) part(6))

```

```

next

```

```

  show ⟨ $\bigwedge j. hi < j \implies j < length\ xs \implies xs'' ! j = xs ! j$ ⟩
  by (metis IH1(4) diff-le-self dual-order.strict-trans2 mset-eq-length part(1) part(5) part(7))

```

```

qed

```

```

qed

```

In the 6th case, we have run quicksort recursively on (lo, p-1, xs'). We show the precondition on the second call on (p+1, hi, xs'')

lemma *quicksort-correct-case6*:

assumes

pre: $\langle \text{quicksort-pre } R \ h \ xs0 \ lo \ hi \ xs \rangle$
and *part*: $\langle \text{partition-spec } R \ h \ xs \ lo \ hi \ xs' \ p \rangle$
and *ifs*: $\langle \neg \ p - \text{Suc } 0 \leq lo \rangle \langle \neg \ hi \leq \text{Suc } p \rangle$
and *IH1*: $\langle \text{quicksort-post } R \ h \ lo \ (p - \text{Suc } 0) \ xs' \ xs'' \rangle$
shows $\langle \text{quicksort-pre } R \ h \ xs0 \ (\text{Suc } p) \ hi \ xs'' \rangle$

proof –

First boilerplate code step: 'unfold' the HOL definitions in the assumptions and convert them to Pure

have *pre*: $\langle lo \leq hi \rangle \langle hi < \text{length } xs \rangle \langle \text{mset } xs0 = \text{mset } xs \rangle$
using *pre* **by** (*auto simp add: quicksort-pre-def*)
have *part*: $\langle \text{mset } xs' = \text{mset } xs \rangle \text{ True}$
 $\langle \text{isPartition-map } R \ h \ xs' \ lo \ hi \ p \rangle \langle lo \leq p \rangle \langle p \leq hi \rangle$
 $\langle \bigwedge i. i < lo \implies xs!i = xs!i \rangle \langle \bigwedge i. \llbracket hi < i; i < \text{length } xs \rrbracket \implies xs!i = xs!i \rangle$
using *part* **by** (*auto simp add: partition-spec-def*)
have *IH1*: $\langle \text{mset } xs'' = \text{mset } xs' \rangle \langle \text{sorted-sublist-map } R \ h \ xs'' \ lo \ (p - \text{Suc } 0) \rangle$
 $\langle \bigwedge i. i < lo \implies xs''!i = xs!i \rangle \langle \bigwedge j. \llbracket p - \text{Suc } 0 < j; j < \text{length } xs \rrbracket \implies xs''!j = xs!j \rangle$
using *IH1* **by** (*auto simp add: quicksort-post-def*)

show *?thesis*

unfolding *quicksort-pre-def*

proof (*intro conjI*)

show $\langle \text{Suc } p \leq hi \rangle$

using *ifs*(2) **by** *linarith*

show $\langle hi < \text{length } xs'' \rangle$

using *IH1*(1) *mset-eq-length part*(1) *pre*(2) **by** *fastforce*

show $\langle \text{mset } xs'' = \text{mset } xs0 \rangle$

using *pre*(3) *part*(1) *IH1*(1) **by** (*auto dest: mset-eq-setD*)

qed

qed

In the 7th (and last) case, we have run quicksort recursively on (lo, p-1, xs'). We show the postcondition on the second call on (p+1, hi, xs'')

lemma *quicksort-correct-case7*:

assumes *trans*: $\langle \bigwedge x \ y \ z. \llbracket R \ (h \ x) \ (h \ y); R \ (h \ y) \ (h \ z) \rrbracket \implies R \ (h \ x) \ (h \ z) \rangle$ **and** *lin*: $\langle \bigwedge x \ y. x \neq y \implies R \ (h \ x) \ (h \ y) \vee R \ (h \ y) \ (h \ x) \rangle$

and *pre*: $\langle \text{quicksort-pre } R \ h \ xs0 \ lo \ hi \ xs \rangle$
and *part*: $\langle \text{partition-spec } R \ h \ xs \ lo \ hi \ xs' \ p \rangle$
and *ifs*: $\langle \neg \ p - \text{Suc } 0 \leq lo \rangle \langle \neg \ hi \leq \text{Suc } p \rangle$
and *IH1'*: $\langle \text{quicksort-post } R \ h \ lo \ (p - \text{Suc } 0) \ xs' \ xs'' \rangle$
and *IH2'*: $\langle \text{quicksort-post } R \ h \ (\text{Suc } p) \ hi \ xs'' \ xs''' \rangle$
shows $\langle \text{quicksort-post } R \ h \ lo \ hi \ xs \ xs''' \rangle$

proof –

First boilerplate code step: 'unfold' the HOL definitions in the assumptions and convert them to Pure

have *pre*: $\langle lo \leq hi \rangle \langle hi < \text{length } xs \rangle$
using *pre* **by** (*auto simp add: quicksort-pre-def*)
have *part*: $\langle \text{mset } xs' = \text{mset } xs \rangle \text{ True}$
 $\langle \text{isPartition-map } R \ h \ xs' \ lo \ hi \ p \rangle \langle lo \leq p \rangle \langle p \leq hi \rangle$
 $\langle \bigwedge i. i < lo \implies xs!i = xs!i \rangle \langle \bigwedge i. \llbracket hi < i; i < \text{length } xs \rrbracket \implies xs!i = xs!i \rangle$
using *part* **by** (*auto simp add: partition-spec-def*)
have *IH1*: $\langle \text{mset } xs'' = \text{mset } xs' \rangle \langle \text{sorted-sublist-map } R \ h \ xs'' \ lo \ (p - \text{Suc } 0) \rangle$

```

  <math>\langle \bigwedge i. i < lo \implies xs''!i = xs!i \rangle \langle \bigwedge j. \llbracket p - Suc\ 0 < j; j < length\ xs \rrbracket \implies xs''!j = xs!j \rangle
  \text{using IH1' by (auto simp add: quicksort-post-def)}
  \text{note IH1-perm} = \text{quicksort-post-set[OF IH1']}
  \text{have IH2: } \langle mset\ xs''' = mset\ xs'' \rangle \langle \text{sorted-sublist-map } R\ h\ xs''' (Suc\ p)\ hi \rangle
  <math>\langle \bigwedge i. i < Suc\ p \implies xs'''!i = xs''!i \rangle \langle \bigwedge j. \llbracket hi < j; j < length\ xs'' \rrbracket \implies xs'''!j = xs''!j \rangle
  \text{using IH2' by (auto simp add: quicksort-post-def)}
  \text{note IH2-perm} = \text{quicksort-post-set[OF IH2']}

```

We still have a partition after the first call (same as in case 5)

```

  \text{have still-partition1: } \langle isPartition-map\ R\ h\ xs''\ lo\ hi\ p \rangle
  \text{proof (intro isPartition-wrtI)}
  \text{fix } i\ \text{assume } \langle lo \leq i \rangle \langle i < p \rangle

```

Obtain the position $posI$ where $xs'' ! i$ was stored in xs' .

```

  \text{have } \langle xs''!i \in set\ (sublist\ xs''\ lo\ (p - Suc\ 0)) \rangle
  \text{by (metis (no-types, lifting) IH1(1) Suc-leI Suc-pred } \langle i < p \rangle \langle lo \leq i \rangle \text{le-less-trans less-imp-diff-less}
  \text{mset-eq-length not-le not-less-zero part(1) part(5) pre(2) sublist-el')}
  \text{then have } \langle xs''!i \in set\ (sublist\ xs'\ lo\ (p - Suc\ 0)) \rangle
  \text{by (metis IH1-perm ifs(1) le-less-trans less-imp-diff-less mset-eq-length nat-le-linear part(1) part(5) pre(2))}
  \text{then have } \langle \exists posI. lo \leq posI \wedge posI \leq p - Suc\ 0 \wedge xs''!i = xs!posI \rangle
  \text{proof - - sledgehammer}
  \text{have } p - Suc\ 0 < length\ xs
  \text{by (meson diff-le-self le-less-trans part(5) pre(2))}
  \text{then show ?thesis}
  \text{by (metis (no-types) } \langle xs''!i \in set\ (sublist\ xs'\ lo\ (p - Suc\ 0)) \rangle \text{ifs(1) mset-eq-length nat-le-linear}
  \text{part(1) sublist-el')}
  \text{qed}
  \text{then obtain } posI :: nat\ \text{where } PosI: \langle lo \leq posI \rangle \langle posI \leq p - Suc\ 0 \rangle \langle xs''!i = xs!posI \rangle \text{by blast}
  \text{then show } \langle R\ (h\ (xs''!i))\ (h\ (xs''!p)) \rangle
  \text{by (metis (no-types, lifting) IH1(4) } \langle i < p \rangle \text{diff-Suc-less isPartition-wrt-def le-less-trans}
  \text{mset-eq-length not-le not-less-eq part(1) part(3) part(5) pre(2) zero-less-Suc)}
  \text{next}
  \text{fix } j\ \text{assume } \langle p < j \rangle \langle j \leq hi \rangle
  \text{then show } \langle R\ (h\ (xs''!p))\ (h\ (xs''!j)) \rangle

```

This holds because this part hasn't changed

```

  \text{by (smt IH1(4) add-diff-cancel-left' add-diff-inverse-nat diff-Suc-eq-diff-pred diff-le-self ifs(1) isPar-}
  \text{tition-wrt-def le-less-Suc-eq less-le-trans mset-eq-length nat-less-le part(1) part(3) part(4) plus-1-eq-Suc}
  \text{pre(2))}
  \text{qed}

```

We still have a partition after the second call (similar as in case 3)

```

  \text{have still-partition2: } \langle isPartition-map\ R\ h\ xs'''\ lo\ hi\ p \rangle
  \text{proof (intro isPartition-wrtI)}
  \text{fix } i\ \text{assume } \langle lo \leq i \rangle \langle i < p \rangle
  \text{show } \langle R\ (h\ (xs'''!i))\ (h\ (xs'''!p)) \rangle

```

This holds because this part hasn't changed

```

  \text{using IH2(3) } \langle i < p \rangle \langle lo \leq i \rangle \text{isPartition-wrt-def still-partition1 by fastforce}
  \text{next}
  \text{fix } j\ \text{assume } \langle p < j \rangle \langle j \leq hi \rangle

```

Obtain the position $posJ$ where $xs''' ! j$ was stored in xs''' .

```

have ⟨ $xs''!j \in \text{set } (\text{sublist } xs''' (Suc\ p)\ hi)$ ⟩
  by (metis IH1(1) IH2(1) Suc-leI ⟨ $j \leq hi$ ⟩ ⟨ $p < j$ ⟩ ifs(2) nat-le-linear part(1) pre(2) size-mset
sublist-el')
then have ⟨ $xs''!j \in \text{set } (\text{sublist } xs'' (Suc\ p)\ hi)$ ⟩
  by (metis IH1(1) IH2-perm ifs(2) mset-eq-length nat-le-linear part(1) pre(2))
then have ⟨ $\exists\ posJ. Suc\ p \leq posJ \wedge posJ \leq hi \wedge xs''!j = xs''!posJ$ ⟩
  by (metis IH1(1) ifs(2) mset-eq-length nat-le-linear part(1) pre(2) sublist-el')
then obtain  $posJ :: nat$  where  $PosJ: \langle Suc\ p \leq posJ \rangle \langle posJ \leq hi \rangle \langle xs''!j = xs''!posJ \rangle$  by blast

then show ⟨ $R\ (h\ (xs'''!\ p))\ (h\ (xs'''!\ j))$ ⟩
proof – — sledgehammer
  have  $\forall\ n\ na\ as\ p. (p\ (as!\ na::'a)\ (as!\ posJ) \vee posJ \leq na) \vee \neg\ isPartition-wrt\ p\ as\ n\ hi\ na$ 
    by (metis (no-types) PosJ(2) isPartition-wrt-def not-less)
  then show ?thesis
    by (metis IH2(3) PosJ(1) PosJ(3) lessI not-less-eq-eq still-partition1)
qed
qed

```

We have that the lower part is sorted after the first recursive call

```
note sorted-lower1 = IH1(2)
```

We show that it is still sorted after the second call.

```

have sorted-lower2: ⟨sorted-sublist-map R h xs''' lo (p–Suc 0)⟩
proof –
  show ?thesis
    using sorted-lower1 apply (rule sorted-wrt-lower-sublist-still-sorted)
    subgoal by (rule part)
    subgoal
      using IH1(1) mset-eq-length part(1) part(5) pre(2) by fastforce
    subgoal
      by (simp add: IH2(3))
    subgoal
      by (metis IH2(1) size-mset)
    done
qed

```

The second IH gives us the the upper list is sorted after the second recursive call

```
note sorted-upper2 = IH2(2)
```

Finally, we have to show that the entire list is sorted after the second recursive call.

```

have sorted-middle: ⟨sorted-sublist-map R h xs''' lo hi⟩
proof –
  show ?thesis
    apply (rule merge-sorted-map-partitions[where  $p=p$ ])
    subgoal by (rule trans)
    subgoal by (rule still-partition2)
    subgoal by (rule sorted-lower2)
    subgoal by (rule sorted-upper2)
    subgoal using pre(1) by auto
    subgoal by (simp add: part(4))
    subgoal by (simp add: part(5))
    subgoal by (metis IH1(1) IH2(1) part(1) pre(2) size-mset)
    done
qed

```

```

show ?thesis
proof (intro quicksort-postI)
  show ⟨mset xs''' = mset xs⟩
  by (simp add: IH1(1) IH2(1) part(1))
next
  show ⟨sorted-sublist-map R h xs''' lo hi⟩
  by (rule sorted-middle)
next
  show ⟨ $\bigwedge i. i < lo \implies xs''' ! i = xs ! i$ ⟩
  using IH1(3) IH2(3) part(4) part(6) by auto
next
  show ⟨ $\bigwedge j. hi < j \implies j < length\ xs \implies xs''' ! j = xs ! j$ ⟩
  by (metis IH1(1) IH1(4) IH2(4) diff-le-self ifs(2) le-SucI less-le-trans nat-le-eq-or-lt not-less
part(1) part(7) size-mset)
qed

```

qed

We can now show the correctness of the abstract quicksort procedure, using the refinement framework and the above case lemmas.

lemma *quicksort-correct*:

assumes *trans*: ⟨ $\bigwedge x y z. \llbracket R(h x)(h y); R(h y)(h z) \rrbracket \implies R(h x)(h z)$ ⟩ **and** *lin*: ⟨ $\bigwedge x y. x \neq y \implies R(h x)(h y) \vee R(h y)(h x)$ ⟩

and *Pre*: ⟨ $lo0 \leq hi0 \wedge hi0 < length\ xs0$ ⟩

shows ⟨ $quicksort\ R\ h\ (lo0, hi0, xs0) \leq \Downarrow Id\ (SPEC(\lambda xs. quicksort\ post\ R\ h\ lo0\ hi0\ xs0\ xs))$ ⟩

proof –

have *wf*: ⟨ $wf\ (measure\ (\lambda(lo, hi, xs). Suc\ hi - lo))$ ⟩

by *auto*

define *pre* **where** ⟨ $pre = (\lambda(lo, hi, xs). quicksort\ pre\ R\ h\ xs0\ lo\ hi\ xs)$ ⟩

define *post* **where** ⟨ $post = (\lambda(lo, hi, xs). quicksort\ post\ R\ h\ lo\ hi\ xs)$ ⟩

have *pre*: ⟨ $pre\ (lo0, hi0, xs0)$ ⟩

unfolding *quicksort-pre-def pre-def* **by** (simp add: *Pre*)

We first generalize the goal a over all states.

have ⟨ $WB\text{-}Sort.\ quicksort\ R\ h\ (lo0, hi0, xs0) \leq \Downarrow Id\ (SPEC\ (post\ (lo0, hi0, xs0)))$ ⟩

unfolding *quicksort-def prod.case*

apply (rule *RECT-rule*)

apply (rule *refine-mono*)

apply (rule *wf*)

apply (rule *pre*)

subgoal *premises IH for f x*

apply (rule *refine-vcg ASSERT-leI*)

unfolding *pre-def post-def*

subgoal — First premise (assertion) for partition

using *IH(2)* **by** (simp add: *quicksort-pre-def pre-def*)

subgoal — Second premise (assertion) for partition

using *IH(2)* **by** (simp add: *quicksort-pre-def pre-def*)

subgoal

using *IH(2)* **by** (auto simp add: *quicksort-pre-def pre-def dest: mset-eq-setD*)

Termination case: $p - (1::'c) \leq lo'$ and $hi' \leq p + (1::'c)$; directly show postcondition

subgoal **unfolding** *partition-spec-def* **by** (auto dest: *mset-eq-setD*)

subgoal — Postcondition (after partition)

apply –

using $IH(2)$ **unfolding** *pre-def* **apply** (*simp, elim conjE, split prod.splits*)
using *trans lin* **apply** (*rule quicksort-correct-case1*) **by** *auto*

Case $p - (1::'c) \leq lo'$ and $hi' < p + (1::'c)$ (Only second recursive call)

subgoal
apply (*rule IH(1)[THEN order-trans]*)

Show that the invariant holds for the second recursive call

subgoal
using $IH(2)$ **unfolding** *pre-def* **apply** (*simp, elim conjE, split prod.splits*)
apply (*rule quicksort-correct-case2*) **by** *auto*

Wellfoundedness (easy)

subgoal by (*auto simp add: quicksort-pre-def partition-spec-def*)

Show that the postcondition holds

subgoal
apply (*simp add: Misc.subset-Collect-conv post-def, intro allI impI, elim conjE*)
using *trans lin* **apply** (*rule quicksort-correct-case3*)
using $IH(2)$ **unfolding** *pre-def* **by** *auto*
done

Case: At least the first recursive call

subgoal
apply (*rule IH(1)[THEN order-trans]*)

Show that the precondition holds for the first recursive call

subgoal
using $IH(2)$ **unfolding** *pre-def post-def* **apply** (*simp, elim conjE, split prod.splits*) **apply** *auto*
apply (*rule quicksort-correct-case4*) **by** *auto*

Wellfoundedness for first recursive call (easy)

subgoal by (*auto simp add: quicksort-pre-def partition-spec-def*)

Simplify some refinement suff...

apply (*simp add: Misc.subset-Collect-conv ASSERT-leI, intro allI impI conjI, elim conjE*)
apply (*rule ASSERT-leI*)
apply (*simp-all add: Misc.subset-Collect-conv ASSERT-leI*)
subgoal unfolding *quicksort-post-def pre-def post-def* **by** (*auto dest: mset-eq-setD*)

Only the first recursive call: show postcondition

subgoal
using *trans lin* **apply** (*rule quicksort-correct-case5*)
using $IH(2)$ **unfolding** *pre-def post-def* **by** *auto*

apply (*rule ASSERT-leI*)
subgoal unfolding *quicksort-post-def pre-def post-def* **by** (*auto dest: mset-eq-setD*)

Both recursive calls.

subgoal
apply (*rule IH(1)[THEN order-trans]*)

Show precondition for second recursive call (after the first call)

```

subgoal
  unfolding pre-def post-def
  apply auto
  apply (rule quicksort-correct-case6)
  using IH(2) unfolding pre-def post-def by auto

```

Wellfoundedness for second recursive call (easy)

```

subgoal by (auto simp add: quicksort-pre-def partition-spec-def)

```

Show that the postcondition holds (after both recursive calls)

```

subgoal
  apply (simp add: Misc.subset-Collect-conv, intro allI impI, elim conjE)
  using trans lin apply (rule quicksort-correct-case7)
  using IH(2) unfolding pre-def post-def by auto
done
done
done
done

```

Finally, apply the generalized lemma to show the thesis.

```

then show ?thesis unfolding post-def by auto
qed

```

definition *partition-main-inv* :: $\langle ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ list} \Rightarrow (\text{nat} \times \text{nat} \times 'a \text{ list}) \Rightarrow \text{bool} \rangle$ **where**

```

   $\langle \text{partition-main-inv } R \ h \ lo \ hi \ xs0 \ p \equiv$ 
     $\text{case } p \text{ of } (i,j,xs) \Rightarrow$ 
       $j < \text{length } xs \wedge j \leq hi \wedge i < \text{length } xs \wedge lo \leq i \wedge i \leq j \wedge \text{mset } xs = \text{mset } xs0 \wedge$ 
       $(\forall k. k \geq lo \wedge k < i \longrightarrow R \ (h \ (xs!k)) \ (h \ (xs!hi))) \wedge$  — All elements from  $lo$  to  $i - (1::'c)$  are smaller than the pivot
       $(\forall k. k \geq i \wedge k < j \longrightarrow R \ (h \ (xs!hi)) \ (h \ (xs!k))) \wedge$  — All elements from  $i$  to  $j - (1::'c)$  are greater than the pivot
       $(\forall k. k < lo \longrightarrow xs!k = xs0!k) \wedge$  — Everything below  $lo$  is unchanged
       $(\forall k. k \geq j \wedge k < \text{length } xs \longrightarrow xs!k = xs0!k)$  — All elements from  $j$  are unchanged (including everything above  $hi$ )
     $\rangle$ 

```

The main part of the partition function. The pivot is assumed to be the last element. This is exactly the "Lomuto partition scheme" partition function from Wikipedia.

definition *partition-main* :: $\langle ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ list} \Rightarrow ('a \text{ list} \times \text{nat}) \text{ nres} \rangle$ **where**

```

   $\langle \text{partition-main } R \ h \ lo \ hi \ xs0 = \text{do } \{$ 
     $\text{ASSERT}(hi < \text{length } xs0);$ 
     $\text{pivot} \leftarrow \text{RETURN } (h \ (xs0 ! hi));$ 
     $(i,j,xs) \leftarrow \text{WHILE}_T^{\text{partition-main-inv } R \ h \ lo \ hi \ xs0}$  — We loop from  $j = lo$  to  $j = hi - (1::'c)$ .
     $(\lambda(i,j,xs). j < hi)$ 
     $(\lambda(i,j,xs). \text{do } \{$ 
       $\text{ASSERT}(i < \text{length } xs \wedge j < \text{length } xs);$ 
       $\text{if } R \ (h \ (xs!j)) \ \text{pivot}$ 
     $\})$ 
   $\rangle$ 

```

```

    then RETURN (i+1, j+1, swap xs i j)
    else RETURN (i, j+1, xs)
  })
  (lo, lo, xs0); — i and j are both initialized to lo
  ASSERT(i < length xs ∧ j = hi ∧ lo ≤ i ∧ hi < length xs ∧ mset xs = mset xs0);
  RETURN (swap xs i hi, i)
}›

```

lemma *partition-main-correct*:

```

assumes bounds: ⟨hi < length xs⟩ ⟨lo ≤ hi⟩ and
  trans: ⟨∧ x y z. [R (h x) (h y); R (h y) (h z)] ⇒ R (h x) (h z)⟩ and lin: ⟨∧ x y. R (h x) (h y) ∨ R
(h y) (h x)⟩
shows ⟨partition-main R h lo hi xs ≤ SPEC(λ(xs', p). mset xs = mset xs' ∧
  lo ≤ p ∧ p ≤ hi ∧ isPartition-map R h xs' lo hi p ∧ (∀ i. i < lo → xs' !i = xs !i) ∧ (∀ i. hi < i ∧ i < length
  xs' → xs' !i = xs !i))⟩

```

proof —

```

have K: ⟨b ≤ hi − Suc n ⇒ n > 0 ⇒ Suc n ≤ hi ⇒ Suc b ≤ hi − n⟩ for b hi n
  by auto
have L: ⟨∼ R (h x) (h y) ⇒ R (h y) (h x)⟩ for x y — Corollary of linearity
  using assms by blast
have M: ⟨a < Suc b ≡ a = b ∨ a < b⟩ for a b
  by linarith
have N: ⟨(a::nat) ≤ b ≡ a = b ∨ a < b⟩ for a b
  by arith

```

show *?thesis*

```

unfolding partition-main-def choose-pivot-def
apply (refine-vcg WHILEIT-rule[where R = ⟨measure(λ(i,j,xs). hi − j)⟩])
subgoal using assms by blast — We feed our assumption to the assertion
subgoal by auto — WF
subgoal — Invariant holds before the first iteration
  unfolding partition-main-inv-def
  using assms apply simp by linarith
subgoal unfolding partition-main-inv-def by simp
subgoal unfolding partition-main-inv-def by simp
subgoal
  unfolding partition-main-inv-def
  apply (auto dest: mset-eq-length)
  done
subgoal unfolding partition-main-inv-def by (auto dest: mset-eq-length)
subgoal
  unfolding partition-main-inv-def apply (auto dest: mset-eq-length)
  by (metis L M mset-eq-length nat-le-eq-or-lt)

```

```

subgoal unfolding partition-main-inv-def by simp — assertions, etc
subgoal unfolding partition-main-inv-def by simp
subgoal unfolding partition-main-inv-def by (auto dest: mset-eq-length)
subgoal unfolding partition-main-inv-def by simp
subgoal unfolding partition-main-inv-def by (auto dest: mset-eq-length)
subgoal unfolding partition-main-inv-def by (auto dest: mset-eq-length)
subgoal unfolding partition-main-inv-def by (auto dest: mset-eq-length)
subgoal unfolding partition-main-inv-def by simp
subgoal unfolding partition-main-inv-def by simp

```


subgoal — After the last iteration, we have a partitioning! :-)
unfolding *partition-main-inv-def* **by** (*auto simp add: isPartition-wrt-def*)
subgoal — And the lower out-of-bounds parts of the list haven't been changed
unfolding *partition-main-inv-def* **by** *auto*
subgoal — And the upper out-of-bounds parts of the list haven't been changed
unfolding *partition-main-inv-def* **by** *auto*
done
qed

definition *partition-between* :: $\langle 'b \Rightarrow 'b \Rightarrow \text{bool} \rangle \Rightarrow \langle 'a \Rightarrow 'b \rangle \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ list} \Rightarrow ('a \text{ list} \times \text{nat})$
nres **where**

$\langle \text{partition-between } R \ h \ lo \ hi \ xs0 = \text{do} \{$
 $\ \ \ \text{ASSERT}(hi < \text{length } xs0 \wedge lo \leq hi);$
 $\ \ \ k \leftarrow \text{choose-pivot } R \ h \ xs0 \ lo \ hi; \text{ — choice of pivot}$
 $\ \ \ \text{ASSERT}(k < \text{length } xs0);$
 $\ \ \ xs \leftarrow \text{RETURN } (\text{swap } xs0 \ k \ hi); \text{ — move the pivot to the last position, before we start the actual}$
loop
 $\ \ \ \text{ASSERT}(\text{length } xs = \text{length } xs0);$
 $\ \ \ \text{partition-main } R \ h \ lo \ hi \ xs$
 $\ \ \ \}$

lemma *partition-between-correct*:

assumes $\langle hi < \text{length } xs \rangle$ **and** $\langle lo \leq hi \rangle$ **and**
 $\langle \bigwedge x \ y \ z. \llbracket R \ (h \ x) \ (h \ y); R \ (h \ y) \ (h \ z) \rrbracket \implies R \ (h \ x) \ (h \ z) \rangle$ **and** $\langle \bigwedge x \ y. R \ (h \ x) \ (h \ y) \vee R \ (h \ y) \ (h \ x) \rangle$
shows $\langle \text{partition-between } R \ h \ lo \ hi \ xs \leq \text{SPEC}(\text{uncurry } (\text{partition-spec } R \ h \ xs \ lo \ hi)) \rangle$

proof —

have $K: \langle b \leq hi - \text{Suc } n \implies n > 0 \implies \text{Suc } n \leq hi \implies \text{Suc } b \leq hi - n \rangle$ **for** $b \ hi \ n$
by *auto*

show *?thesis*

unfolding *partition-between-def choose-pivot-def*
apply (*refine-vcg partition-main-correct*)
using *assms* **apply** (*auto dest: mset-eq-length simp add: partition-spec-def*)
by (*metis dual-order.strict-trans2 less-imp-not-eq2 mset-eq-length swap-nth*)

qed

We use the median of the first, the middle, and the last element.

definition *choose-pivot3* **where**

$\langle \text{choose-pivot3 } R \ h \ xs \ lo \ (hi::\text{nat}) = \text{do} \{$
 $\ \ \ \text{ASSERT}(lo < \text{length } xs);$
 $\ \ \ \text{ASSERT}(hi < \text{length } xs);$
 $\ \ \ \text{let } k' = (hi - lo) \ \text{div } 2;$
 $\ \ \ \text{let } k = lo + k';$
 $\ \ \ \text{ASSERT}(k < \text{length } xs);$
 $\ \ \ \text{let } start = h \ (xs \ ! \ lo);$
 $\ \ \ \text{let } mid = h \ (xs \ ! \ k);$
 $\ \ \ \text{let } end = h \ (xs \ ! \ hi);$
 $\ \ \ \text{if } (R \ start \ mid \wedge R \ mid \ end) \vee (R \ end \ mid \wedge R \ mid \ start) \text{ then } \text{RETURN } k$
 $\ \ \ \text{else if } (R \ start \ end \wedge R \ end \ mid) \vee (R \ mid \ end \wedge R \ end \ start) \text{ then } \text{RETURN } hi$
 $\ \ \ \text{else } \text{RETURN } lo$
 $\ \ \ \}$

— We only have to show that this procedure yields a valid index between *lo* and *hi*.

lemma *choose-pivot3-choose-pivot*:
assumes $\langle lo < length\ xs \ \langle hi < length\ xs \ \langle hi \geq lo \rangle$
shows $\langle choose\text{-}pivot3\ R\ h\ xs\ lo\ hi \leq \Downarrow Id\ (choose\text{-}pivot\ R\ h\ xs\ lo\ hi) \rangle$
unfolding *choose-pivot3-def choose-pivot-def*
using *assms* **by** (*auto intro!*: *ASSERT-leI simp: Let-def*)

The refined partion function: We use the above pivot function and fold instead of non-deterministic iteration.

definition *partition-between-ref*
 $:: \langle ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow nat \Rightarrow nat \Rightarrow 'a\ list \Rightarrow ('a\ list \times nat)\ nres \rangle$
where
 $\langle partition\text{-}between\text{-}ref\ R\ h\ lo\ hi\ xs0 = do \{$
 $\quad ASSERT(hi < length\ xs0 \wedge hi < length\ xs0 \wedge lo \leq hi);$
 $\quad k \leftarrow choose\text{-}pivot3\ R\ h\ xs0\ lo\ hi; \text{ --- choice of pivot}$
 $\quad ASSERT(k < length\ xs0);$
 $\quad xs \leftarrow RETURN\ (swap\ xs0\ k\ hi); \text{ --- move the pivot to the last position, before we start the actual}$
 loop
 $\quad ASSERT(length\ xs = length\ xs0);$
 $\quad partition\text{-}main\ R\ h\ lo\ hi\ xs$
 $\} \rangle$

lemma *partition-main-ref'*:
 $\langle partition\text{-}main\ R\ h\ lo\ hi\ xs$
 $\leq \Downarrow ((\lambda a\ b\ c\ d.\ Id)\ a\ b\ c\ d)\ (partition\text{-}main\ R\ h\ lo\ hi\ xs) \rangle$
by *auto*

lemma *Down-id-eq*:
 $\langle \Downarrow Id\ x = x \rangle$
by *auto*

lemma *partition-between-ref-partition-between*:
 $\langle partition\text{-}between\text{-}ref\ R\ h\ lo\ hi\ xs \leq (partition\text{-}between\ R\ h\ lo\ hi\ xs) \rangle$

proof –
have *swap*: $\langle (swap\ xs\ k\ hi, swap\ xs\ ka\ hi) \in Id \rangle$ **if** $\langle k = ka \rangle$
for *k ka*
using *that* **by** *auto*
have [*refine0*]: $\langle (h\ (xsa\ !\ hi), h\ (xsa\ !\ hi)) \in Id \rangle$
if $\langle (xsa, xsa) \in Id \rangle$
for *xsa xsa*
using *that* **by** *auto*

show *?thesis*
apply (*subst* (2) *Down-id-eq[symmetric]*)
unfolding *partition-between-ref-def*
 $\quad partition\text{-}between\text{-}def$
 $\quad OP\text{-}def$
apply (*refine-vcg choose-pivot3-choose-pivot swap partition-main-correct*)
subgoal **by** *auto*
subgoal **by** *auto*
subgoal **by** *auto*
subgoal **by** *auto*
subgoal **by** *auto*

```

subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
by (auto intro: Refine-Basic.Id-refine dest: mset-eq-length)
qed

```

Technical lemma for sepref

```

lemma partition-between-ref-partition-between':
  ⟨(uncurry2 (partition-between-ref R h), uncurry2 (partition-between R h)) ∈
    (nat-rel ×r nat-rel) ×r ⟨Id⟩list-rel →f ⟨⟨Id⟩list-rel ×r nat-rel⟩nres-rel⟩
  by (intro frefI nres-relI)
  (auto intro: partition-between-ref-partition-between)

```

Example instantiation for pivot

```

definition choose-pivot3-impl where
  ⟨choose-pivot3-impl = choose-pivot3 (≤) id⟩

```

```

lemma partition-between-ref-correct:
  assumes trans: ⟨∧ x y z. [R (h x) (h y); R (h y) (h z)] ⇒ R (h x) (h z)⟩ and lin: ⟨∧ x y. R (h x)
    (h y) ∨ R (h y) (h x)⟩
  and bounds: ⟨hi < length xs⟩ ⟨lo ≤ hi⟩
  shows ⟨partition-between-ref R h lo hi xs ≤ SPEC (uncurry (partition-spec R h xs lo hi))⟩
proof -
  show ?thesis
  apply (rule partition-between-ref-partition-between[THEN order-trans])
  using bounds apply (rule partition-between-correct[where h=h])
  subgoal by (rule trans)
  subgoal by (rule lin)
  done
qed

```

Refined quicksort algorithm: We use the refined partition function.

```

definition quicksort-ref :: ⟨- ⇒ - ⇒ nat × nat × 'a list ⇒ 'a list nres⟩ where
  ⟨quicksort-ref R h = (λ(lo,hi,xs0).
    do {
      RECT (λf (lo,hi,xs). do {
        ASSERT(lo ≤ hi ∧ hi < length xs0 ∧ mset xs = mset xs0);
        (xs, p) ← partition-between-ref R h lo hi xs; — This is the refined partition function. Note that we
        need the premises (trans,lin,bounds) here.
        ASSERT(mset xs = mset xs0 ∧ p ≥ lo ∧ p < length xs0);
        xs ← (if p-1 ≤ lo then RETURN xs else f (lo, p-1, xs));
        ASSERT(mset xs = mset xs0);
        if hi ≤ p+1 then RETURN xs else f (p+1, hi, xs)
      }) (lo,hi,xs0)
    })⟩

```

```

lemma fref-to-Down-curry2:
  ⟨(uncurry2 f, uncurry2 g) ∈ [P]f A → ⟨B⟩nres-rel ⇒
    (∧ x x' y y' z z'. P ((x', y'), z') ⇒ (((x, y), z), ((x', y'), z')) ∈ A ⇒
      f x y z ≤↓ B (g x' y' z'))⟩
  unfolding fref-def uncurry-def nres-rel-def

```

by *auto*

lemma *fref-to-Down-curry*:

$\langle (f, g) \in [P]_f A \rightarrow \langle B \rangle \text{nres-rel} \implies$
 $(\bigwedge x x'. P x' \implies (x, x') \in A \implies$
 $f x \leq \Downarrow B (g x')) \rangle$

unfolding *fref-def uncurry-def nres-rel-def*

by *auto*

lemma *quicksort-ref-quicksort*:

assumes *bounds*: $\langle hi < \text{length } xs \rangle \langle lo \leq hi \rangle$ **and**

trans: $\langle \bigwedge x y z. \llbracket R (h x) (h y); R (h y) (h z) \rrbracket \implies R (h x) (h z) \rangle$ **and** *lin*: $\langle \bigwedge x y. R (h x) (h y) \vee R$
 $(h y) (h x) \rangle$

shows $\langle \text{quicksort-ref } R \ h \ x0 \leq \Downarrow Id (\text{quicksort } R \ h \ x0) \rangle$

proof –

have *wf*: $\langle wf (\text{measure } (\lambda(lo, hi, xs). \text{Suc } hi - lo)) \rangle$

by *auto*

have *pre*: $\langle x0 = x0' \implies (x0, x0') \in Id \times_r Id \times_r \langle Id \rangle \text{list-rel} \rangle$ **for** $x0 \ x0' :: \langle \text{nat} \times \text{nat} \times 'b \text{ list} \rangle$

by *auto*

have [*refine0*]: $\langle (x1e = x1d) \implies (x1e, x1d) \in Id \rangle$ **for** $x1e \ x1d :: \langle 'b \text{ list} \rangle$

by *auto*

show *?thesis*

unfolding *quicksort-def quicksort-ref-def*

apply (*refine-vcg pre partition-between-ref-partition-between*'[*THEN fref-to-Down-curry2*])

First assertion (premise for partition)

subgoal

by *auto*

First assertion (premise for partition)

subgoal

by *auto*

subgoal

by (*auto dest: mset-eq-length*)

subgoal

by (*auto dest: mset-eq-length mset-eq-setD*)

Correctness of the concrete partition function

subgoal

apply (*simp, rule partition-between-ref-correct*)

subgoal by (*rule trans*)

subgoal by (*rule lin*)

subgoal by *auto* — first premise

subgoal by *auto* — second premise

done

subgoal

by (*auto dest: mset-eq-length mset-eq-setD*)

subgoal by (*auto simp: partition-spec-def isPartition-wrt-def*)

subgoal by (*auto simp: partition-spec-def isPartition-wrt-def dest: mset-eq-length*)

subgoal

by (*auto dest: mset-eq-length mset-eq-setD*)

subgoal

by (auto dest: mset-eq-length mset-eq-setD)
 subgoal
 by (auto dest: mset-eq-length mset-eq-setD)
 subgoal
 by (auto dest: mset-eq-length mset-eq-setD)

by simp+
 qed

— Sort the entire list

definition *full-quicksort* **where**

$\langle \text{full-quicksort } R \ h \ xs \equiv \text{if } xs = [] \text{ then RETURN } xs \text{ else quicksort } R \ h \ (0, \text{length } xs - 1, xs) \rangle$

definition *full-quicksort-ref* **where**

$\langle \text{full-quicksort-ref } R \ h \ xs \equiv$
 if *List.null* *xs* then RETURN *xs*
 else *quicksort-ref* *R* *h* (0, *length xs* - 1, *xs*) \rangle

definition *full-quicksort-impl* :: $\langle \text{nat list} \Rightarrow \text{nat list nres} \rangle$ **where**

$\langle \text{full-quicksort-impl } xs = \text{full-quicksort-ref } (\leq) \ \text{id } xs \rangle$

lemma *full-quicksort-ref-full-quicksort*:

assumes *trans*: $\langle \bigwedge x \ y \ z. \llbracket R \ (h \ x) \ (h \ y); R \ (h \ y) \ (h \ z) \rrbracket \Longrightarrow R \ (h \ x) \ (h \ z) \rangle$ **and** *lin*: $\langle \bigwedge x \ y. R \ (h \ x) \ (h \ y) \vee R \ (h \ y) \ (h \ x) \rangle$

shows $\langle (\text{full-quicksort-ref } R \ h, \text{full-quicksort } R \ h) \in \langle \text{Id} \rangle \text{list-rel} \rightarrow_f \langle \text{Id} \rangle \text{list-rel} \rangle \text{nres-rel} \rangle$

proof —

show *?thesis*

unfolding *full-quicksort-ref-def* *full-quicksort-def*

apply (*intro* *frefI* *nres-relI*)

apply (*auto* *intro!*: *quicksort-ref-quicksort*[*unfolded* *Down-id-eq*] *simp*: *List.null-def*)

subgoal **by** (*rule* *trans*)

subgoal **using** *lin* **by** *blast*

done

qed

lemma *sublist-entire*:

$\langle \text{sublist } xs \ 0 \ (\text{length } xs - 1) = xs \rangle$

by (*simp* *add*: *sublist-def*)

lemma *sorted-sublist-wrt-entire*:

assumes $\langle \text{sorted-sublist-wrt } R \ xs \ 0 \ (\text{length } xs - 1) \rangle$

shows $\langle \text{sorted-wrt } R \ xs \rangle$

proof —

have $\langle \text{sorted-wrt } R \ (\text{sublist } xs \ 0 \ (\text{length } xs - 1)) \rangle$

using *assms* **by** (*simp* *add*: *sorted-sublist-wrt-def*)

then show *?thesis*

by (*metis* *sublist-entire*)

qed

lemma *sorted-sublist-map-entire*:

assumes $\langle \text{sorted-sublist-map } R \ h \ xs \ 0 \ (\text{length } xs - 1) \rangle$

shows $\langle \text{sorted-wrt } (\lambda x \ y. R \ (h \ x) \ (h \ y)) \ xs \rangle$

proof –
show *?thesis*
using *assms* **by** (*rule sorted-sublist-wrt-entire*)
qed

Final correctness lemma

theorem *full-quicksort-correct-sorted*:

assumes
trans: $\langle \bigwedge x y z. \llbracket R(h x) (h y); R(h y) (h z) \rrbracket \implies R(h x) (h z) \rangle$ **and** *lin*: $\langle \bigwedge x y. x \neq y \implies R(h x) (h y) \vee R(h y) (h x) \rangle$
shows $\langle \text{full-quicksort } R \ h \ xs \leq \Downarrow \text{Id } (\text{SPEC}(\lambda xs'. \text{mset } xs' = \text{mset } xs \wedge \text{sorted-wrt } (\lambda x y. R(h x) (h y)) \ xs')) \rangle$

proof –
show *?thesis*
unfolding *full-quicksort-def*
apply (*refine-vcg*)
subgoal **by** *simp* — case $xs = []$
subgoal **by** *simp* — case $xs = []$

apply (*rule quicksort-correct[THEN order-trans]*)
subgoal **by** (*rule trans*)
subgoal **by** (*rule lin*)
subgoal **by** *linarith*
subgoal **by** *simp*

apply (*simp add: Misc.subset-Collect-conv, intro allI impI conjI*)
subgoal
by (*auto simp add: quicksort-post-def*)
subgoal
apply (*rule sorted-sublist-map-entire*)
by (*auto simp add: quicksort-post-def dest: mset-eq-length*)
done

qed

lemma *full-quicksort-correct*:

assumes
trans: $\langle \bigwedge x y z. \llbracket R(h x) (h y); R(h y) (h z) \rrbracket \implies R(h x) (h z) \rangle$ **and**
lin: $\langle \bigwedge x y. R(h x) (h y) \vee R(h y) (h x) \rangle$
shows $\langle \text{full-quicksort } R \ h \ xs \leq \Downarrow \text{Id } (\text{SPEC}(\lambda xs'. \text{mset } xs' = \text{mset } xs)) \rangle$
by (*rule order-trans[OF full-quicksort-correct-sorted]*)
(use assms in auto)

end

theory *More-Loops*

imports

Refine-Monadic.Refine-While
Refine-Monadic.Refine-Foreach
HOL-Library.Rewrite

begin

3.3 More Theorem about Loops

Most theorem below have a counterpart in the Refinement Framework that is weaker (by missing assertions for example that are critical for code generation).

lemma *Down-id-eq*:

$\langle \Downarrow Id\ x = x \rangle$
by *auto*

lemma *while-upt-while-direct1*:

$b \geq a \implies$
do {
 $(-, \sigma) \leftarrow WHILE_T (FOREACH\text{-}cond\ c) (\lambda x. do \{ ASSERT (FOREACH\text{-}cond\ c\ x); FOREACH\text{-}body\ f\ x \})$
 $([a..<b], \sigma);$
 $RETURN\ \sigma$
 $\} \leq do \{$
 $(-, \sigma) \leftarrow WHILE_T (\lambda(i, x). i < b \wedge c\ x) (\lambda(i, x). do \{ ASSERT (i < b); \sigma' \leftarrow f\ i\ x; RETURN (i+1, \sigma')$
 $\}) (a, \sigma);$
 $RETURN\ \sigma$
 $\}$
apply (*rewrite at* $\langle - \leq \sqsupset \rangle$ *Down-id-eq[symmetric]*)
apply (*refine-vcg* *WHILE_T-refine[where* $R = \langle \{((l, x'), (i::nat, x::'a)). x = x' \wedge i \leq b \wedge i \geq a \wedge l = drop\ (i-a)\ [a..<b]\} \rangle$ *]*)
subgoal by *auto*
subgoal by (*auto simp: FOREACH-cond-def*)
subgoal by (*auto simp: FOREACH-body-def intro!: bind-refine[OF Id-refine]*)
subgoal by *auto*
done

lemma *while-upt-while-direct2*:

$b \geq a \implies$
do {
 $(-, \sigma) \leftarrow WHILE_T (FOREACH\text{-}cond\ c) (\lambda x. do \{ ASSERT (FOREACH\text{-}cond\ c\ x); FOREACH\text{-}body\ f\ x \})$
 $([a..<b], \sigma);$
 $RETURN\ \sigma$
 $\} \geq do \{$
 $(-, \sigma) \leftarrow WHILE_T (\lambda(i, x). i < b \wedge c\ x) (\lambda(i, x). do \{ ASSERT (i < b); \sigma' \leftarrow f\ i\ x; RETURN (i+1, \sigma')$
 $\}) (a, \sigma);$
 $RETURN\ \sigma$
 $\}$
apply (*rewrite at* $\langle - \leq \sqsupset \rangle$ *Down-id-eq[symmetric]*)
apply (*refine-vcg* *WHILE_T-refine[where* $R = \langle \{((i::nat, x::'a), (l, x')). x = x' \wedge i \leq b \wedge i \geq a \wedge l = drop\ (i-a)\ [a..<b]\} \rangle$ *]*)
subgoal by *auto*
subgoal by (*auto simp: FOREACH-cond-def*)
subgoal by (*auto simp: FOREACH-body-def intro!: bind-refine[OF Id-refine]*)
subgoal by (*auto simp: FOREACH-body-def intro!: bind-refine[OF Id-refine]*)
subgoal by *auto*
done

lemma *while-upt-while-direct*:

$b \geq a \implies$
do {
 $(-, \sigma) \leftarrow WHILE_T (FOREACH\text{-}cond\ c) (\lambda x. do \{ ASSERT (FOREACH\text{-}cond\ c\ x); FOREACH\text{-}body\ f\ x \})$
 $([a..<b], \sigma);$
 $RETURN\ \sigma$
 $\} = do \{$

```

  (-,σ) ← WHILET (λ(i, x). i < b ∧ c x) (λ(i, x). do {ASSERT (i < b); σ' ← f i x; RETURN (i+1, σ')
}) (a, σ);
  RETURN σ
}
using while-upt-while-direct1[of a b] while-upt-while-direct2[of a b]
unfolding order-eq-iff by fast

```

lemma *while-nfoldli*:

```

do {
  (-,σ) ← WHILET (FOREACH-cond c) (λx. do {ASSERT (FOREACH-cond c x); FOREACH-body
f x}) (l, σ);
  RETURN σ
} ≤ nfoldli l c f σ
apply (induct l arbitrary: σ)
apply (subst WHILET-unfold)
apply (simp add: FOREACH-cond-def)

```

```

apply (subst WHILET-unfold)
apply (auto
simp: FOREACH-cond-def FOREACH-body-def
intro: bind-mono Refine-Basic.bind-mono(1))

```

done

lemma *nfoldli-while*: $nfoldli\ l\ c\ f\ \sigma$

```

≤
(WHILETI
(FOREACH-cond c) (λx. do {ASSERT (FOREACH-cond c x); FOREACH-body f x}) (l, σ)
)
≫
(λ(-, σ). RETURN σ)

```

proof (induct l arbitrary: σ)

case Nil **thus** ?case **by** (subst WHILE_T-unfold) (auto simp: FOREACH-cond-def)

next

case (Cons x ls)

show ?case

proof (cases c σ)

case False **thus** ?thesis

apply (subst WHILE_T-unfold)

unfolding FOREACH-cond-def

by simp

next

case [simp]: True

from Cons **show** ?thesis

apply (subst WHILE_T-unfold)

unfolding FOREACH-cond-def FOREACH-body-def

apply clarsimp

apply (rule Refine-Basic.bind-mono)

apply simp-all

done

qed

qed

lemma *while-eq-nfoldli*: do {

```

  (-,σ) ← WHILET (FOREACH-cond c) (λx. do {ASSERT (FOREACH-cond c x); FOREACH-body
f x}) (l, σ);
  RETURN σ

```

```

} = nfoldli l c f σ

```



```

apply (rule antisym)
apply (rule while-nfoldli)
apply (rule order-trans[OF nfoldli-while[where  $I=\lambda\cdot$ . True]])
apply (simp add: WHILET-def)
done

```

end

```

theory PAC-Specification
  imports PAC-More-Poly
begin

```

4 Specification of the PAC checker

4.1 Ideals

```

type-synonym int-poly =  $\langle$ int mpoly $\rangle$ 
definition polynomial-bool ::  $\langle$ int-poly set $\rangle$  where
   $\langle$ polynomial-bool =  $(\lambda c. \text{Var } c \wedge 2 - \text{Var } c)$  ‘UNIV $\rangle$ 

```

```

definition pac-ideal where
   $\langle$ pac-ideal  $A \equiv \text{ideal } (A \cup \text{polynomial-bool})$  $\rangle$ 

```

```

lemma X2-X-in-pac-ideal:
   $\langle$ Var  $c \wedge 2 - \text{Var } c \in \text{pac-ideal } A$  $\rangle$ 
  unfolding polynomial-bool-def pac-ideal-def
  by (auto intro: ideal.span-base)

```

```

lemma pac-idealI1 [intro]:
   $\langle$  $p \in A \implies p \in \text{pac-ideal } A$  $\rangle$ 
  unfolding pac-ideal-def
  by (auto intro: ideal.span-base)

```

```

lemma pac-idealI2 [intro]:
   $\langle$  $p \in \text{ideal } A \implies p \in \text{pac-ideal } A$  $\rangle$ 
  using ideal.span-subspace-induct pac-ideal-def by blast

```

```

lemma pac-idealI3 [intro]:
   $\langle$  $p \in \text{ideal } A \implies p * q \in \text{pac-ideal } A$  $\rangle$ 
  by (metis ideal.span-scale mult commute pac-idealI2)

```

```

lemma pac-ideal-Xsq2-iff:
   $\langle$  $\text{Var } c \wedge 2 \in \text{pac-ideal } A \iff \text{Var } c \in \text{pac-ideal } A$  $\rangle$ 
  unfolding pac-ideal-def
  apply (subst (2) ideal.span-add-eq[symmetric, OF X2-X-in-pac-ideal[of c, unfolded pac-ideal-def]])
  apply auto
  done

```

```

lemma diff-in-polynomial-bool-pac-idealI:
  assumes a1:  $p \in \text{pac-ideal } A$ 
  assumes a2:  $p - p' \in \text{More-Modules.ideal polynomial-bool}$ 
  shows  $\langle$  $p' \in \text{pac-ideal } A$  $\rangle$ 
proof –
  have insert  $p$  polynomial-bool  $\subseteq$  pac-ideal  $A$ 
  using a1 unfolding pac-ideal-def by (meson ideal.span-superset insert-subset le-sup-iff)

```

then show *?thesis*
using *a2 unfolding pac-ideal-def* **by** (*metis (no-types) ideal.eq-span-insert-eq ideal.span-subset-spanI ideal.span-superset insert-subset subsetD*)
qed

lemma *diff-in-polynomial-bool-pac-idealI2*:
assumes *a1: p ∈ A*
assumes *a2: p - p' ∈ More-Modules.ideal polynomial-bool*
shows $\langle p' \in \text{pac-ideal } A \rangle$
using *diff-in-polynomial-bool-pac-idealI[OF - assms(2), of A] assms(1)*
by (*auto simp: ideal.span-base*)

lemma *pac-ideal-alt-def*:
 $\langle \text{pac-ideal } A = \text{ideal } (A \cup \text{ideal polynomial-bool}) \rangle$
unfolding *pac-ideal-def*
by (*meson ideal.span-eq ideal.span-mono ideal.span-superset le-sup-iff subset-trans sup-ge2*)

The equality on ideals is restricted to polynomials whose variable appear in the set of ideals.
The function restrict sets:

definition *restricted-ideal-to* **where**
 $\langle \text{restricted-ideal-to } B A = \{p \in A. \text{vars } p \subseteq B\} \rangle$

abbreviation *restricted-ideal-to_I* **where**
 $\langle \text{restricted-ideal-to}_I B A \equiv \text{restricted-ideal-to } B (\text{pac-ideal } (\text{set-mset } A)) \rangle$

abbreviation *restricted-ideal-to_V* **where**
 $\langle \text{restricted-ideal-to}_V B \equiv \text{restricted-ideal-to } (\bigcup (\text{vars } \text{'set-mset } B)) \rangle$

abbreviation *restricted-ideal-to_{V I}* **where**
 $\langle \text{restricted-ideal-to}_{V I} B A \equiv \text{restricted-ideal-to } (\bigcup (\text{vars } \text{'set-mset } B)) (\text{pac-ideal } (\text{set-mset } A)) \rangle$

lemma *restricted-idealI*:
 $\langle p \in \text{pac-ideal } (\text{set-mset } A) \implies \text{vars } p \subseteq C \implies p \in \text{restricted-ideal-to}_I C A \rangle$
unfolding *restricted-ideal-to-def*
by *auto*

lemma *pac-ideal-insert-already-in*:
 $\langle pq \in \text{pac-ideal } (\text{set-mset } A) \implies \text{pac-ideal } (\text{insert } pq (\text{set-mset } A)) = \text{pac-ideal } (\text{set-mset } A) \rangle$
by (*auto simp: pac-ideal-alt-def ideal.span-insert-idI*)

lemma *pac-ideal-add*:
 $\langle p \in \# A \implies q \in \# A \implies p + q \in \text{pac-ideal } (\text{set-mset } A) \rangle$
by (*simp add: ideal.span-add ideal.span-base pac-ideal-def*)

lemma *pac-ideal-mult*:
 $\langle p \in \# A \implies p * q \in \text{pac-ideal } (\text{set-mset } A) \rangle$
by (*simp add: ideal.span-base pac-idealI3*)

lemma *pac-ideal-mono*:
 $\langle A \subseteq B \implies \text{pac-ideal } A \subseteq \text{pac-ideal } B \rangle$
using *ideal.span-mono[of A U -> B U ->]*
by (*auto simp: pac-ideal-def intro: ideal.span-mono*)

4.2 PAC Format

The PAC format contains three kind of steps:

- **add** that adds up two polynomials that are known.
- **mult** that multiply a known polynomial with another one.
- **del** that removes a polynomial that cannot be reused anymore.

To model the simplification that happens, we add the $p - p' \in \text{polynomial-bool}$ stating that p and p' are equivalent.

type-synonym $\text{pac-st} = \langle (\text{nat set} \times \text{int-poly multiset}) \rangle$

inductive $\text{PAC-Format} :: \langle \text{pac-st} \Rightarrow \text{pac-st} \Rightarrow \text{bool} \rangle$ **where**

add:

$\langle \text{PAC-Format } (\mathcal{V}, A) (\mathcal{V}, \text{add-mset } p' A) \rangle$

if

$\langle p \in \# A \rangle \langle q \in \# A \rangle$
 $\langle p+q - p' \in \text{ideal polynomial-bool} \rangle$
 $\langle \text{vars } p' \subseteq \mathcal{V} \rangle \mid$

mult:

$\langle \text{PAC-Format } (\mathcal{V}, A) (\mathcal{V}, \text{add-mset } p' A) \rangle$

if

$\langle p \in \# A \rangle$
 $\langle p*q - p' \in \text{ideal polynomial-bool} \rangle$
 $\langle \text{vars } p' \subseteq \mathcal{V} \rangle$
 $\langle \text{vars } q \subseteq \mathcal{V} \rangle \mid$

del:

$\langle p \in \# A \implies \text{PAC-Format } (\mathcal{V}, A) (\mathcal{V}, A - \{\#p\#}) \rangle \mid$

extend-pos:

$\langle \text{PAC-Format } (\mathcal{V}, A) (\mathcal{V} \cup \{x' \in \text{vars } (-\text{Var } x + p'), x' \notin \mathcal{V}\}, \text{add-mset } (-\text{Var } x + p') A) \rangle$

if

$\langle (p')^2 - p' \in \text{ideal polynomial-bool} \rangle$
 $\langle \text{vars } p' \subseteq \mathcal{V} \rangle$
 $\langle x \notin \mathcal{V} \rangle$

In the PAC format above, we have a technical condition on the normalisation: $\text{vars } p' \subseteq \text{vars } (p + q)$ is here to ensure that we don't normalise 0 to $(\text{Var } x)^2 - \text{Var } x$ for a new variable x . This is completely obvious for the normalisation process we have in mind when we write the specification, but we must add it explicitly because we are too general.

lemmas $\text{PAC-Format-induct-split} =$

$\text{PAC-Format.induct}[\text{split-format}(\text{complete}), \text{of } V A V' A' \text{ for } V A V' A']$

lemma $\text{PAC-Format-induct}[\text{consumes } 1, \text{case-names add mult del ext}]$:

assumes

$\langle \text{PAC-Format } (\mathcal{V}, A) (\mathcal{V}', A') \rangle$ **and**

cases:

$\langle \bigwedge p q p' A \mathcal{V}. p \in \# A \implies q \in \# A \implies p+q - p' \in \text{ideal polynomial-bool} \implies \text{vars } p' \subseteq \mathcal{V} \implies P$
 $\mathcal{V} A \mathcal{V} (\text{add-mset } p' A) \rangle$

$\langle \bigwedge p q p' A \mathcal{V}. p \in \# A \implies p*q - p' \in \text{ideal polynomial-bool} \implies \text{vars } p' \subseteq \mathcal{V} \implies \text{vars } q \subseteq \mathcal{V} \implies$
 $P \mathcal{V} A \mathcal{V} (\text{add-mset } p' A) \rangle$

$\langle \bigwedge p A \mathcal{V}. p \in \# A \implies P \mathcal{V} A \mathcal{V} (A - \{\#p\#}) \rangle$

$\langle \bigwedge p' x r. \rangle$

$(p')^{\wedge 2} - (p') \in \text{ideal polynomial-bool} \implies \text{vars } p' \subseteq \mathcal{V} \implies$
 $x \notin \mathcal{V} \implies P \mathcal{V} A (\mathcal{V} \cup \{x' \in \text{vars } (p' - \text{Var } x), x' \notin \mathcal{V}\}) (\text{add-mset } (p' - \text{Var } x) A)$

shows

$\langle P \mathcal{V} A \mathcal{V}' A' \rangle$

using *assms(1)* **apply** $-$

by (*induct* $V \equiv \mathcal{V} A \equiv A \mathcal{V}' A'$ *rule: PAC-Format-induct-split*)

(*auto intro: assms(1) cases*)

The theorem below (based on the proof ideal by Manuel Kauers) is the correctness theorem of extensions. Remark that the assumption $\text{vars } q \subseteq \mathcal{V}$ is only used to show that $x' \notin \text{vars } q$.

lemma *extensions-are-safe:*

assumes $\langle x' \in \text{vars } p \rangle$ **and**

$x': \langle x' \notin \mathcal{V} \rangle$ **and**

$\langle \bigcup (\text{vars } \text{'set-mset } A) \subseteq \mathcal{V} \rangle$ **and**

$p\text{-x-coeff}: \langle \text{coeff } p (\text{monomial } (\text{Suc } 0) x') = 1 \rangle$ **and**

$\text{vars-}q: \langle \text{vars } q \subseteq \mathcal{V} \rangle$ **and**

$q: \langle q \in \text{More-Modules.ideal } (\text{insert } p (\text{set-mset } A \cup \text{polynomial-bool})) \rangle$ **and**

leading: $\langle x' \notin \text{vars } (p - \text{Var } x') \rangle$ **and**

diff: $\langle (\text{Var } x' - p)^2 - (\text{Var } x' - p) \in \text{More-Modules.ideal polynomial-bool} \rangle$

shows

$\langle q \in \text{More-Modules.ideal } (\text{set-mset } A \cup \text{polynomial-bool}) \rangle$

proof $-$

define p' **where** $\langle p' \equiv p - \text{Var } x' \rangle$

let $?v = \langle \text{Var } x' :: \text{int mpoly} \rangle$

have $p\text{-}p': \langle p = ?v + p' \rangle$

by (*auto simp: p'-def*)

define q' **where** $\langle q' \equiv \text{Var } x' - p \rangle$

have $q\text{-}q': \langle p = ?v - q' \rangle$

by (*auto simp: q'-def*)

have *diff*: $\langle q'^{\wedge 2} - q' \in \text{More-Modules.ideal polynomial-bool} \rangle$

using *diff unfolding q-q' by auto*

have [*simp*]: $\langle \text{vars } ((\text{Var } c)^2 - \text{Var } c :: \text{int mpoly}) = \{c\} \rangle$ **for** c

apply (*auto simp: vars-def Var-def Var₀-def mpoly.MPoly-inverse keys-def lookup-minus-fun lookup-times-monomial-right single.rep-eq split: if-splits*)

apply (*auto simp: vars-def Var-def Var₀-def mpoly.MPoly-inverse keys-def lookup-minus-fun lookup-times-monomial-right single.rep-eq when-def ac-simps adds-def lookup-plus-fun power2-eq-square times-mpoly.rep-eq minus-mpoly.rep-eq split: if-splits*)

apply (*rule-tac* $x = \langle ?2 :: \text{nat} \Rightarrow_0 \text{nat} \rangle * \text{monomial } (\text{Suc } 0) c$) **in** *exI*)

apply (*auto dest: monomial-0D simp: plus-eq-zero-2 lookup-plus-fun mult-2*)

by (*meson Suc-neq-Zero monomial-0D plus-eq-zero-2*)

have *eq*: $\langle \text{More-Modules.ideal } (\text{insert } p (\text{set-mset } A \cup \text{polynomial-bool})) =$

$\text{More-Modules.ideal } (\text{insert } p (\text{set-mset } A \cup (\lambda c. \text{Var } c^{\wedge 2} - \text{Var } c) \text{' } \{c. c \neq x'\}) \rangle$

(**is** $\langle ?A = ?B \rangle$ **is** $\langle - = \text{More-Modules.ideal } ?\text{trimmed} \rangle$)

proof $-$

let $?C = \langle \text{insert } p (\text{set-mset } A \cup (\lambda c. \text{Var } c^{\wedge 2} - \text{Var } c) \text{' } \{c. c \neq x'\}) \rangle$

let $?D = \langle (\lambda c. \text{Var } c^{\wedge 2} - \text{Var } c) \text{' } \{c. c \neq x'\} \rangle$

have *diff*: $\langle q'^{\wedge 2} - q' \in \text{More-Modules.ideal } ?D \rangle$ (**is** $\langle ?q \in - \rangle$)

proof $-$

obtain $r t$ **where**

$q: \langle ?q = (\sum a \in t. r a * a) \rangle$ **and**

fin-t: $\langle \text{finite } t \rangle$ **and**

$t: \langle t \subseteq \text{polynomial-bool} \rangle$

```

using diff unfolding ideal.span-explicit
by auto
show ?thesis
proof (cases ⟨?v2 - ?v ∉ t⟩)
case True
then show ⟨?thesis⟩
using q fin-t t unfolding ideal.span-explicit
by (auto intro!: exI[of - ⟨t - {?v2 - ?v}⟩] exI[of - r]
simp: polynomial-bool-def sum-diff1)
next
case False
define t' where ⟨t' = t - {?v2 - ?v}⟩
have t-t': ⟨t = insert (?v2 - ?v) t'⟩ and
notin: ⟨?v2 - ?v ∉ t'⟩ and
⟨t' ⊆ (λc. Var c ^ 2 - Var c) ' {c. c ≠ x'}⟩
using False t unfolding t'-def polynomial-bool-def by auto
have mon: ⟨monom (monomial (Suc 0) x') 1 = Var x'⟩
by (auto simp: coeff-def minus-mpoly.rep-eq Var-def Var0-def monom-def
times-mpoly.rep-eq lookup-minus lookup-times-monomial-right mpoly.MPoly-inverse)
then have ⟨∀ a. ∃ g h. r a = ?v * g + h ∧ x' ∉ vars h⟩
using polynomial-split-on-var[of ⟨r -> x'⟩]
by metis
then obtain g h where
r: ⟨r a = ?v * g a + h a⟩ and
x'-h: ⟨x' ∉ vars (h a)⟩ for a
using polynomial-split-on-var[of ⟨r a⟩ x']
by metis
have ⟨?q = ((∑ a ∈ t'. g a * a) + r (?v2 - ?v) * (?v - 1)) * ?v + (∑ a ∈ t'. h a * a)⟩
using fin-t notin unfolding t-t' q r
by (auto simp: field-simps comm-monoid-add-class.sum.distrib
power2-eq-square ideal.scale-left-commute sum-distrib-left)
moreover have ⟨x' ∉ vars ?q⟩
by (metis (no-types, opaque-lifting) Groups.add-ac(2) Un-iff add-diff-cancel-left'
diff-minus-eq-add in-mono leading q'-def semiring-normalization-rules(29)
vars-in-right-only vars-mult)
moreover {
have ⟨x' ∉ (⋃ m ∈ t' - {?v2 - ?v}. vars (h m * m))⟩
using fin-t x'-h vars-mult[of ⟨h ->⟩] ⟨t' ⊆ polynomial-bool⟩
by (auto simp: polynomial-bool-def t-t' elim!: vars-unE)
then have ⟨x' ∉ vars (∑ a ∈ t'. h a * a)⟩
using vars-setsum[of ⟨t'⟩ ⟨λa. h a * a⟩] fin-t x'-h t notin
by (auto simp: t-t')
}
ultimately have ⟨?q = (∑ a ∈ t'. h a * a)⟩
unfolding mon[symmetric]
by (rule polynomial-decomp-alien-var(2)[unfolded])
then show ?thesis
using t fin-t ⟨t' ⊆ (λc. Var c ^ 2 - Var c) ' {c. c ≠ x'}⟩
unfolding ideal.span-explicit t-t'
by auto
qed
qed
have eq1: ⟨More-Modules.ideal (insert p (set-mset A ∪ polynomial-bool)) =
More-Modules.ideal (insert (?v2 - ?v) ?C)⟩
(is ⟨More-Modules.ideal - = More-Modules.ideal (insert - ?C)⟩)

```

```

    by (rule arg-cong[of - - More-Modules.ideal])
      (auto simp: polynomial-bool-def)
  moreover have ⟨ $?v^2 - ?v \in \text{More-Modules.ideal } ?C$ ⟩
  proof -
    have ⟨ $?v - q' \in \text{More-Modules.ideal } ?C$ ⟩
      by (auto simp: q-q' ideal.span-base)
    from ideal.span-scale[OF this, of ⟨ $?v + q' - 1$ ⟩] have ⟨ $(?v - q') * (?v + q' - 1) \in \text{More-Modules.ideal } ?C$ ⟩
  by (auto simp: field-simps)
  moreover have ⟨ $q'^2 - q' \in \text{More-Modules.ideal } ?C$ ⟩
    using diff by (smt Un-insert-right ideal.span-mono insert-subset subsetD sup-ge2)
  ultimately have ⟨ $(?v - q') * (?v + q' - 1) + (q'^2 - q') \in \text{More-Modules.ideal } ?C$ ⟩
    by (rule ideal.span-add)
  moreover have ⟨ $?v^2 - ?v = (?v - q') * (?v + q' - 1) + (q'^2 - q')$ ⟩
    by (auto simp: p'-def q-q' field-simps power2-eq-square)
  ultimately show ?thesis by simp
qed
ultimately show ?thesis
  using ideal.span-insert-idI by blast
qed

have ⟨ $n < m \implies n > 0 \implies \exists q. ?v^n = ?v + q * (?v^2 - ?v)$ ⟩ for n m :: nat
proof (induction m arbitrary: n)
  case 0
  then show ?case by auto
next
  case (Suc m n) note IH = this(1-)
  consider
    ⟨n < m⟩ |
    ⟨m = n⟩ ⟨n > 1⟩ |
    ⟨n = 1⟩
  using IH
  by (cases ⟨n < m⟩; cases n) auto
  then show ?case
proof cases
  case 1
  then show ?thesis using IH by auto
next
  case 2
  have eq: ⟨ $?v^n = ((?v :: \text{int mpoly}) ^ (n-2)) * (?v^2 - ?v) + ?v^{n-1}$ ⟩
    using 2 by (auto simp: field-simps power-eq-if
      ideal.scale-right-diff-distrib)
  obtain q where
    q: ⟨ $?v^{n-1} = ?v + q * (?v^2 - ?v)$ ⟩
    using IH(1)[of ⟨n-1⟩] 2
    by auto
  show ?thesis
    using q unfolding eq
    by (auto intro!: exI[of - ⟨Var x' ^ (n - 2) + q⟩] simp: distrib-right)
next
  case 3
  then show ⟨?thesis⟩
    by auto
qed
qed

```

```

obtain  $r\ t$  where
   $q$ :  $\langle q = (\sum a \in t. r\ a * a) \rangle$  and
   $fin\text{-}t$ :  $\langle finite\ t \rangle$  and
   $t$ :  $\langle t \subseteq ?trimmed \rangle$ 
  using  $q$  unfolding  $eq$  unfolding  $ideal.span\text{-}explicit$ 
  by  $auto$ 

define  $t'$  where  $\langle t' \equiv t - \{p\} \rangle$ 
have  $t'$ :  $\langle t = (if\ p \in t\ then\ insert\ p\ t'\ else\ t') \rangle$  and
   $t''[simp]$ :  $\langle p \notin t' \rangle$ 
  unfolding  $t'\text{-}def$  by  $auto$ 
show  $?thesis$ 
proof ( $cases\ \langle r\ p = 0 \vee p \notin t \rangle$ )
  case  $True$ 
  have
     $q$ :  $\langle q = (\sum a \in t'. r\ a * a) \rangle$  and
     $fin\text{-}t$ :  $\langle finite\ t' \rangle$  and
     $t$ :  $\langle t' \subseteq set\text{-}mset\ A \cup polynomial\text{-}bool \rangle$ 
    using  $q\ fin\text{-}t\ t\ True\ t''$ 
    apply ( $subst\ (asm)\ t'$ )
    apply ( $auto\ intro$ :  $sum.cong\ simp$ :  $sum.insert\text{-}remove\ t'\text{-}def$ )
    using  $q\ fin\text{-}t\ t\ True\ t''$ 
    apply ( $auto\ intro$ :  $sum.cong\ simp$ :  $sum.insert\text{-}remove\ t'\text{-}def\ polynomial\text{-}bool\text{-}def$ )
    done
  then show  $?thesis$ 
  by ( $auto\ simp$ :  $ideal.span\text{-}explicit$ )
next
  case  $False$ 
  then have  $\langle r\ p \neq 0 \rangle$  and  $\langle p \in t \rangle$ 
  by  $auto$ 
  then have  $t$ :  $\langle t = insert\ p\ t' \rangle$ 
  by ( $auto\ simp$ :  $t'\text{-}def$ )

have  $\langle x' \notin vars\ (-\ p') \rangle$ 
  using  $leading\ p'\text{-}def\ vars\text{-}in\text{-}right\text{-}only$  by  $fastforce$ 
have  $mon$ :  $\langle monom\ (monomial\ (Suc\ 0)\ x')\ 1 = Var\ x' \rangle$ 
  by ( $auto\ simp$ :  $coeff\text{-}def\ minus\text{-}mpoly.rep\text{-}eq\ Var\text{-}def\ Var_0\text{-}def\ monom\text{-}def$ 
     $times\text{-}mpoly.rep\text{-}eq\ lookup\text{-}minus\ lookup\text{-}times\text{-}monomial\text{-}right\ mpoly.MPoly\text{-}inverse$ )
then have  $\langle \forall a. \exists g\ h. r\ a = (?v + p') * g + h \wedge x' \notin vars\ h \rangle$ 
  using  $polynomial\text{-}split\text{-}on\text{-}var2[of\ x'\ \langle -p' \rangle\ \langle r \text{-} \rangle]\ \langle x' \notin vars\ (-\ p') \rangle$ 
  by ( $metis\ diff\text{-}minus\text{-}eq\text{-}add$ )
then obtain  $g\ h$  where
   $r$ :  $\langle r\ a = p * g\ a + h\ a \rangle$  and
   $x'\text{-}h$ :  $\langle x' \notin vars\ (h\ a) \rangle$  for  $a$ 
  using  $polynomial\text{-}split\text{-}on\text{-}var2[of\ x'\ p'\ \langle r\ a \rangle]$  unfolding  $p\text{-}p'[symmetric]$ 
  by  $metis$ 

have  $ISABLELLE\text{-}come\text{-}on$ :  $\langle a * (p * g\ a) = p * (a * g\ a) \rangle$  for  $a$ 
  by  $auto$ 
have  $q1$ :  $\langle q = p * (\sum a \in t'. g\ a * a) + (\sum a \in t'. h\ a * a) + p * r\ p \rangle$ 
  ( $is\ \langle - = - + ?NOx' + - \rangle$ )
  using  $fin\text{-}t\ t''$  unfolding  $q\ t\ ISABLELLE\text{-}come\text{-}on\ r$ 

```

```

apply (subst semiring-class.distrib-right)+
apply (auto simp: comm-monoid-add-class.sum.distrib semigroup-mult-class.mult.assoc
  ISABLL-come-on simp flip: semiring-0-class.sum-distrib-right
  semiring-0-class.sum-distrib-left)
by (auto simp: field-simps)
also have  $\langle \dots = ((\sum a \in t'. g a * a) + r p) * p + (\sum a \in t'. h a * a) \rangle$ 
by (auto simp: field-simps)
finally have q-decomp:  $\langle q = ((\sum a \in t'. g a * a) + r p) * p + (\sum a \in t'. h a * a) \rangle$ 
  (is  $\langle q = ?X * p + ?NOx' \rangle$ ).

have [iff]:  $\langle \text{monomial (Suc 0) } c = 0 - \text{monomial (Suc 0) } c = \text{False} \rangle$  for c
by (metis One-nat-def diff-is-0-eq' le-eq-less-or-eq less-Suc-eq-le monomial-0-iff single-diff zero-neq-one)
have  $\langle x \in t' \implies x' \in \text{vars } x \implies \text{False} \rangle$  for x
using  $\langle t \subseteq ?\text{trimmed} \rangle$  t assms(2,3)
apply (auto simp: polynomial-bool-def dest!: multi-member-split)
apply (frule set-rev-mp)
apply assumption
apply (auto dest!: multi-member-split)
done
then have  $\langle x' \notin (\bigcup m \in t'. \text{vars } (h m * m)) \rangle$ 
using fin-t x'-h vars-mult[of <h ->]
by (auto simp: t elim!: vars-unE)
then have  $\langle x' \notin \text{vars } ?NOx' \rangle$ 
using vars-setsum[of <t'> <\lambda a. h a * a>] fin-t x'-h
by (auto simp: t)

moreover {
  have  $\langle x' \notin \text{vars } p' \rangle$ 
using assms(7)
unfolding p'-def
by auto
then have  $\langle x' \notin \text{vars } (h p * p') \rangle$ 
using vars-mult[of <h p> p'] x'-h
by auto
}
ultimately have
 $\langle x' \notin \text{vars } q \rangle$ 
 $\langle x' \notin \text{vars } ?NOx' \rangle$ 
 $\langle x' \notin \text{vars } p' \rangle$ 
using x' vars-q vars-add[of <h p * p'> <\sum a \in t'. h a * a>] x'-h
  leading p'-def
by auto
then have  $\langle ?X = 0 \rangle$  and q-decomp:  $\langle q = ?NOx' \rangle$ 
unfolding mon[symmetric] p-p'
using polynomial-decomp-alien-var2[OF q-decomp[unfolded p-p' mon[symmetric]]]
by auto

then have  $\langle r p = (\sum a \in t'. (- g a) * a) \rangle$ 
  (is  $\langle - = ?CL \rangle$ )
unfolding add.assoc add-eq-0-iff equation-minus-iff
by (auto simp: sum-negf ac-simps)

then have q2:  $\langle q = (\sum a \in t'. a * (r a - p * g a)) \rangle$ 

```



```

using fin-t unfolding q
apply (auto simp: t r q
  comm-monoid-add-class.sum.distrib[symmetric]
  sum-distrib-left
  sum-distrib-right
  left-diff-distrib
  intro!: sum.cong)
apply (auto simp: field-simps)
done
then show <?thesis>
  using t fin-t <t ⊆ ?trimmed> unfolding ideal.span-explicit
  by (auto intro!: exI[of - t^] exI[of - <λa. r a - p * g a>]
    simp: field-simps polynomial-bool-def)
qed
qed

```

```

lemma extensions-are-safe-uminus:
  assumes <x' ∈ vars p> and
    x': <x' ∉ V> and
    <⋃ (vars ' set-mset A) ⊆ V> and
    p-x-coeff: <coeff p (monomial (Suc 0) x') = -1> and
    vars-q: <vars q ⊆ V> and
    q: <q ∈ More-Modules.ideal (insert p (set-mset A ∪ polynomial-bool))> and
    leading: <x' ∉ vars (p + Var x')> and
    diff: <(Var x' + p)^2 - (Var x' + p) ∈ More-Modules.ideal polynomial-bool>
  shows
    <q ∈ More-Modules.ideal (set-mset A ∪ polynomial-bool)>
proof -
  have <q ∈ More-Modules.ideal (insert (- p) (set-mset A ∪ polynomial-bool))>
    by (metis ideal.span-breakdown-eq minus-mult-minus q)

  then show ?thesis
    using extensions-are-safe[of x' <-p> V A q] assms
    using vars-in-right-only by force
qed

```

This is the correctness theorem of a PAC step: no polynomials are added to the ideal.

```

lemma vars-subst-in-left-only:
  <x ∉ vars p ⇒ x ∈ vars (p - Var x)> for p :: <int mpoly>
  by (metis One-nat-def Var.abs-eq Var0-def group-eq-aux monom.abs-eq mult-numeral-1 polynomial-decomp-alien-var(1)
    zero-neq-numeral)

```

```

lemma vars-subst-in-left-only-diff-iff:
  fixes p :: <int mpoly>
  assumes <x ∉ vars p>
  shows <vars (p - Var x) = insert x (vars p)>
proof -
  have <⋀xa. x ∉ vars p ⇒ xa ∈ vars (p - Var x) ⇒ xa ∉ vars p ⇒ xa = x>
    by (metis (no-types, opaque-lifting) diff-0-right diff-minus-eq-add empty-iff in-vars-addE insert-iff
      keys-single minus-diff-eq monom-one mult.right-neutral one-neq-zero single-zero
      vars-monom-keys vars-mult-Var vars-uminus)
  moreover have <⋀xa. x ∉ vars p ⇒ xa ∈ vars p ⇒ xa ∈ vars (p - Var x)>
    by (metis add.inverse-inverse diff-minus-eq-add empty-iff insert-iff keys-single minus-diff-eq
      monom-one mult.right-neutral one-neq-zero single-zero vars-in-right-only vars-monom-keys
      vars-mult-Var vars-uminus)

```

ultimately show *?thesis*
using *assms*
by (*auto simp: vars-subst-in-left-only*)
qed

lemma *vars-subst-in-left-only-iff*:
 $\langle x \notin \text{vars } p \implies \text{vars } (p + \text{Var } x) = \text{insert } x (\text{vars } p) \rangle$ **for** $p :: \langle \text{int mpoly} \rangle$
using *vars-subst-in-left-only-diff-iff*[*of x <-p>*]
by (*metis diff-0 diff-diff-add vars-uminus*)

lemma *coeff-add-right-notin*:
 $\langle x \notin \text{vars } p \implies \text{MPoly-Type.coeff } (\text{Var } x - p) (\text{monomial } (\text{Suc } 0) x) = 1 \rangle$
apply (*auto simp flip: coeff-minus simp: not-in-vars-coeff0*)
by (*simp add: MPoly-Type.coeff-def Var.rep-eq Var₀-def*)

lemma *coeff-add-left-notin*:
 $\langle x \notin \text{vars } p \implies \text{MPoly-Type.coeff } (p - \text{Var } x) (\text{monomial } (\text{Suc } 0) x) = -1 \rangle$ **for** $p :: \langle \text{int mpoly} \rangle$
apply (*auto simp flip: coeff-minus simp: not-in-vars-coeff0*)
by (*simp add: MPoly-Type.coeff-def Var.rep-eq Var₀-def*)

lemma *ideal-insert-polynomial-bool-swap*: $\langle r - s \in \text{ideal polynomial-bool} \implies$
 $\text{More-Modules.ideal } (\text{insert } r (A \cup \text{polynomial-bool})) = \text{More-Modules.ideal } (\text{insert } s (A \cup \text{polynomial-bool})) \rangle$
apply *auto*
using *ideal.eq-span-insert-eq ideal.span-mono sup-ge2* **apply** *blast+*
done

lemma *PAC-Format-subset-ideal*:
 $\langle \text{PAC-Format } (\mathcal{V}, A) (\mathcal{V}', B) \implies \bigcup (\text{vars } \langle \text{set-mset } A \rangle \subseteq \mathcal{V} \implies$
 $\text{restricted-ideal-to}_I \mathcal{V} B \subseteq \text{restricted-ideal-to}_I \mathcal{V} A \wedge \mathcal{V} \subseteq \mathcal{V}' \wedge \bigcup (\text{vars } \langle \text{set-mset } B \rangle \subseteq \mathcal{V}') \rangle$
unfolding *restricted-ideal-to-def*
apply (*induction rule:PAC-Format-induct*)
subgoal for $p \ q \ pq \ A \ \mathcal{V}$
using *vars-add*
by (*force simp: ideal.span-add-eq ideal.span-base pac-ideal-insert-already-in*[*OF diff-in-polynomial-bool-pac-idealI*][*of*
 $\langle p + q \rangle \langle \cdot \rangle pq$])
pac-ideal-add
intro!: *diff-in-polynomial-bool-pac-idealI*][*of* $\langle p + q \rangle \langle \cdot \rangle pq$])
subgoal for $p \ q \ pq$
using *vars-mult*[*of p q*]
by (*force simp: ideal.span-add-eq ideal.span-base pac-ideal-mult*
pac-ideal-insert-already-in[*OF diff-in-polynomial-bool-pac-idealI*][*of* $\langle p * q \rangle \langle \cdot \rangle pq$])
subgoal for $p \ A$
using *pac-ideal-mono*[*of* $\langle \text{set-mset } (A - \{\#p\#\}) \rangle \langle \text{set-mset } A \rangle$]
by (*auto dest: in-diffD*)
subgoal for $p \ x' \ r'$
apply (*subgoal-tac* $\langle x' \notin \text{vars } p \rangle$)
using *extensions-are-safe-uminus*[*of* $x' \langle \text{Var } x' + p \rangle \mathcal{V} A$] **unfolding** *pac-ideal-def*
apply (*auto simp: vars-subst-in-left-only coeff-add-left-notin*)
done
done

In general, if deletions are disallowed, then the stronger $B = \text{pac-ideal } A$ holds.

lemma *restricted-ideal-to-restricted-ideal-to_ID*:
 $\langle \text{restricted-ideal-to } \mathcal{V} (\text{set-mset } A) \subseteq \text{restricted-ideal-to}_I \mathcal{V} A \rangle$

by (auto simp add: Collect-disj-eq pac-idealI1 restricted-ideal-to-def)

lemma rtranclp-PAC-Format-subset-ideal:

$\langle \text{rtranclp PAC-Format } (\mathcal{V}, A) (\mathcal{V}', B) \implies \bigcup (\text{vars ' set-mset } A) \subseteq \mathcal{V} \implies$
 $\text{restricted-ideal-to}_I \mathcal{V} B \subseteq \text{restricted-ideal-to}_I \mathcal{V} A \wedge \mathcal{V} \subseteq \mathcal{V}' \wedge \bigcup (\text{vars ' set-mset } B) \subseteq \mathcal{V}' \rangle$

apply (induction rule:rtranclp-induct[of PAC-Format $\langle(-, -)\rangle \langle(-, -)\rangle$, split-format(complete)])

subgoal

by (simp add: restricted-ideal-to-restricted-ideal-to_ID)

subgoal

by (drule PAC-Format-subset-ideal)

(auto simp: restricted-ideal-to-def Collect-mono-iff)

done

end

theory PAC-Map-Rel

imports

Refine-Imperative-HOL.IICF Finite-Map-Multiset

begin

5 Hash-Map for finite mappings

This function declares hash-maps for ('a, 'b) fmap, that are nicer to use especially here where everything is finite.

definition fmap-rel **where**

[to-relAPP]:

fmap-rel K V $\equiv \{(m1, m2).$

$(\forall i j. i \in | \text{fmdom } m2 \longrightarrow (j, i) \in K \longrightarrow (\text{the } (\text{fmlookup } m1 j), \text{the } (\text{fmlookup } m2 i)) \in V) \wedge$

$\text{fset } (\text{fmdom } m1) \subseteq \text{Domain } K \wedge \text{fset } (\text{fmdom } m2) \subseteq \text{Range } K \wedge$

$(\forall i j. (i, j) \in K \longrightarrow j \in | \text{fmdom } m2 \longleftrightarrow i \in | \text{fmdom } m1)\}$

lemma fmap-rel-alt-def:

$\langle K, V \rangle \text{fmap-rel} \equiv$

$\{(m1, m2).$

$(\forall i j. i \in \# \text{dom-m } m2 \longrightarrow$

$(j, i) \in K \longrightarrow (\text{the } (\text{fmlookup } m1 j), \text{the } (\text{fmlookup } m2 i)) \in V) \wedge$

$\text{fset } (\text{fmdom } m1) \subseteq \text{Domain } K \wedge$

$\text{fset } (\text{fmdom } m2) \subseteq \text{Range } K \wedge$

$(\forall i j. (i, j) \in K \longrightarrow (j \in \# \text{dom-m } m2) = (i \in \# \text{dom-m } m1))\}$

,

unfolding fmap-rel-def dom-m-def fmember.rep-eq

by auto

lemma fmdom-empty-fmempty-iff[simp]: $\langle \text{fmdom } m = \{\} \longleftrightarrow m = \text{fmempty} \rangle$

by (metis fmdom-empty fmdrop-fset-fmdom fmdrop-fset-null)

lemma fmap-rel-empty1-simp[simp]:

$(\text{fmempty}, m) \in \langle K, V \rangle \text{fmap-rel} \longleftrightarrow m = \text{fmempty}$

apply (cases $\langle \text{fmdom } m = \{\} \rangle$)

apply (auto simp: fmap-rel-def)[]

by (auto simp add: fmember.rep-eq fmap-rel-def simp del: fmdom-empty-fmempty-iff)

lemma *fmap-rel-empty2-simp*[simp]:
 $(m, \text{fmempty}) \in \langle K, V \rangle \text{fmap-rel} \longleftrightarrow m = \text{fmempty}$
apply (*cases* $\langle \text{fndom } m = \{\|\} \rangle$)
apply (*auto simp: fmap-rel-def*)[]
by (*fastforce simp add: fmember.rep-eq fmap-rel-def simp del: fndom-empty-fmempty-iff*)

sempref-decl-intf $(\text{'k}, \text{'v})$ *f-map* **is** $(\text{'k}, \text{'v})$ *fmap*

lemma [*synth-rules*]: $\llbracket \text{INTF-OF-REL } K \text{ TYPE}(\text{'k}); \text{INTF-OF-REL } V \text{ TYPE}(\text{'v}) \rrbracket$
 $\implies \text{INTF-OF-REL } (\langle K, V \rangle \text{fmap-rel}) \text{ TYPE}((\text{'k}, \text{'v}) \text{f-map})$ **by** *simp*

5.1 Operations

sempref-decl-op *fmap-empty*: *fmempty* :: $\langle K, V \rangle \text{fmap-rel}$.

sempref-decl-op *fmap-is-empty*: $(=) \text{fmempty} :: \langle K, V \rangle \text{fmap-rel} \rightarrow \text{bool-rel}$
apply (*rule fref-ncI*)
apply *parametricity*
apply (*rule fun-relI; auto*)
done

lemma *fmap-rel-fmupd-fmap-rel*:
 $\langle A, B \rangle \in \langle K, R \rangle \text{fmap-rel} \implies (p, p') \in K \implies (q, q') \in R \implies$
 $(\text{fmupd } p \ q \ A, \text{fmupd } p' \ q' \ B) \in \langle K, R \rangle \text{fmap-rel}$
if *single-valued* *K* *single-valued* (K^{-1})
using *that*
unfolding *fmap-rel-alt-def*
apply (*case-tac* $\langle p' \in \# \text{ dom-m } B \rangle$)
apply (*auto simp add: all-conj-distrib IS-RIGHT-UNIQUED dest!: multi-member-split*)
done

sempref-decl-op *fmap-update*: *fmupd* :: $K \rightarrow V \rightarrow \langle K, V \rangle \text{fmap-rel} \rightarrow \langle K, V \rangle \text{fmap-rel}$
where *single-valued* *K* *single-valued* (K^{-1})
apply (*rule fref-ncI*)
apply *parametricity*
apply (*intro fun-relI*)
by (*rule fmap-rel-fmupd-fmap-rel*)

lemma *remove1-mset-eq-add-mset-iff*:
 $\langle \text{remove1-mset } a \ A = \text{add-mset } a \ A' \longleftrightarrow A = \text{add-mset } a \ (\text{add-mset } a \ A') \rangle$
by (*metis add-mset-add-single add-mset-diff-bothsides diff-zero remove1-mset-eqE*)

lemma *fmap-rel-fmdrop-fmap-rel*:
 $\langle \text{fmdrop } p \ A, \text{fmdrop } p' \ B \rangle \in \langle K, R \rangle \text{fmap-rel}$
if *single*: *single-valued* *K* *single-valued* (K^{-1}) **and**
 $H0: \langle A, B \rangle \in \langle K, R \rangle \text{fmap-rel} \langle (p, p') \in K \rangle$
proof –
have *H*: $\langle \bigwedge A a \ j. \forall i. i \in \# \text{ dom-m } B \longrightarrow (\forall j. (j, i) \in K \longrightarrow (\text{the } (\text{fmlookup } A \ j), \text{the } (\text{fmlookup } B \ i)) \in R) \implies \text{remove1-mset } p' \ (\text{dom-m } B) = \text{add-mset } p' \ Aa \implies (j, p') \in K \implies \text{False} \rangle$
by (*metis dom-m-fmdrop fmlookup-drop in-dom-m-lookup-iff union-single-eq-member*)
have *H2*: $\langle \bigwedge i \ A a \ j. (p, p') \in K \implies$

$\forall i. i \in \# \text{ dom-m } B \longrightarrow (\forall j. (j, i) \in K \longrightarrow (\text{the } (\text{fmlookup } A \ j), \text{the } (\text{fmlookup } B \ i)) \in R) \implies$
 $\forall i j. (i, j) \in K \longrightarrow (j \in \# \text{ dom-m } B) = (i \in \# \text{ dom-m } A) \implies$
 $\text{remove1-mset } p' (\text{dom-m } B) = \text{add-mset } i \ Aa \implies$
 $(j, i) \in K \implies$
 $(\text{the } (\text{fmlookup } A \ j), \text{the } (\text{fmlookup } B \ i)) \in R \wedge j \in \# \text{ remove1-mset } p (\text{dom-m } A) \wedge$
 $i \in \# \text{ remove1-mset } p' (\text{dom-m } B)$
 $\langle \wedge i j \ Aa.$
 $(p, p') \in K \implies$
 $\text{single-valued } K \implies$
 $\text{single-valued } (K^{-1}) \implies$
 $\forall i. i \in \# \text{ dom-m } B \longrightarrow (\forall j. (j, i) \in K \longrightarrow (\text{the } (\text{fmlookup } A \ j), \text{the } (\text{fmlookup } B \ i)) \in R) \implies$
 $\text{fset } (\text{fmdom } A) \subseteq \text{Domain } K \implies$
 $\text{fset } (\text{fmdom } B) \subseteq \text{Range } K \implies$
 $\forall i j. (i, j) \in K \longrightarrow (j \in \# \text{ dom-m } B) = (i \in \# \text{ dom-m } A) \implies$
 $(i, j) \in K \implies \text{remove1-mset } p (\text{dom-m } A) = \text{add-mset } i \ Aa \implies j \in \# \text{ remove1-mset } p' (\text{dom-m } B)$
 \rangle
using *single*
by (*metis IS-RIGHT-UNIQUED converse.intros dom-m-fmdrop fmlookup-drop in-dom-m-lookup-iff union-single-eq-member*)
show $\langle \text{fmdrop } p \ A, \text{fmdrop } p' \ B \rangle \in \langle K, R \rangle \text{fmap-rel}$
using *that*
unfolding *fmap-rel-alt-def*
by (*auto simp add: all-conj-distrib IS-RIGHT-UNIQUED dest!: multi-member-split dest: H H2*)
qed

sempref-decl-op *fmap-delete: fmdrop :: K → ⟨K, V⟩fmap-rel → ⟨K, V⟩fmap-rel*
where *single-valued K single-valued (K⁻¹)*
apply (*rule fref-ncI*)
apply *parametricity*
by (*auto simp add: fmap-rel-fmdrop-fmap-rel*)

lemma *fmap-rel-nat-the-fmlookup[intro]:*
 $\langle (A, B) \in \langle S, R \rangle \text{fmap-rel} \implies (p, p') \in S \implies p' \in \# \text{ dom-m } B \implies$
 $(\text{the } (\text{fmlookup } A \ p), \text{the } (\text{fmlookup } B \ p')) \in R \rangle$
by (*auto simp: fmap-rel-alt-def distinct-mset-dom*)

lemma *fmap-rel-in-dom-iff:*
 $\langle (aa, a'a) \in \langle K, V \rangle \text{fmap-rel} \implies$
 $(a, a') \in K \implies$
 $a' \in \# \text{ dom-m } a'a \longleftrightarrow$
 $a \in \# \text{ dom-m } aa \rangle$
unfolding *fmap-rel-alt-def*
by *auto*

lemma *fmap-rel-fmlookup-rel:*
 $\langle (a, a') \in K \implies (aa, a'a) \in \langle K, V \rangle \text{fmap-rel} \implies$
 $(\text{fmlookup } aa \ a, \text{fmlookup } a'a \ a') \in \langle V \rangle \text{option-rel} \rangle$
using *fmap-rel-nat-the-fmlookup[of aa a'a K V a a']*
fmap-rel-in-dom-iff[of aa a'a K V a a']
in-dom-m-lookup-iff[of a' a'a]
in-dom-m-lookup-iff[of a aa]
by (*cases* $\langle a' \in \# \text{ dom-m } a'a \rangle$)
(auto simp del: fmap-rel-nat-the-fmlookup)

```

sempref-decl-op fmap-lookup: fmlookup :: ⟨K, V⟩fmap-rel → K → ⟨V⟩option-rel
  apply (rule fref-ncI)
  apply parametricity
  apply (intro fun-relI)
  apply (rule fmap-rel-fmlookup-rel; assumption)
  done

```

```

lemma in-fdom-alt: k∈#dom-m m ↔ ¬is-None (fmlookup m k)
  apply (auto split: option.split intro: fmdom-notI simp: dom-m-def fmember.rep-eq)
  apply (meson fmdom-notI notin-fset)
  using notin-fset by fastforce

```

```

sempref-decl-op fmap-contains-key: λk m. k∈#dom-m m :: K → ⟨K, V⟩fmap-rel → bool-rel
  unfolding in-fdom-alt
  apply (rule fref-ncI)
  apply parametricity
  apply (rule fmap-rel-fmlookup-rel; assumption)
  done

```

5.2 Patterns

```

lemma pat-fmap-empty[pat-rules]: fmempty ≡ op-fmap-empty by simp

```

```

lemma pat-map-is-empty[pat-rules]:
  (=) $m$fmempty ≡ op-fmap-is-empty$m
  (=) $fmempty$m ≡ op-fmap-is-empty$m
  (=) $(dom-m$m){#} ≡ op-fmap-is-empty$m
  (=) ${#}$$(dom-m$m) ≡ op-fmap-is-empty$m
  unfolding atomize-eq
  by (auto dest: sym)

```

```

lemma op-map-contains-key[pat-rules]:
  (∈#) $ k $ (dom-m$m) ≡ op-fmap-contains-key$k$m
  by (auto intro!: eq-reflection)

```

5.3 Mapping to Normal Hashmaps

```

abbreviation map-of-fmap :: ⟨'k ⇒ 'v option⟩ ⇒ ⟨'k, 'v⟩fmap where
  ⟨map-of-fmap h ≡ Abs-fmap h⟩

```

```

definition map-fmap-rel where
  ⟨map-fmap-rel = br map-of-fmap (λa. finite (dom a))⟩

```

```

lemma fmdrop-set-None:
  ⟨(op-map-delete, fmdrop) ∈ Id → map-fmap-rel → map-fmap-rel⟩
  apply (auto simp: map-fmap-rel-def br-def)
  apply (subst fmdrop.abs-eq)
  apply (auto simp: eq-onp-def fmap.Abs-fmap-inject
    map-drop-def map-filter-finite
    intro!: ext)
  apply (auto simp: map-filter-def)
  done

```

```

lemma map-upd-fmupd:
  ⟨(op-map-update, fmupd) ∈ Id → Id → map-fmap-rel → map-fmap-rel⟩

```

```

apply (auto simp: map-fmap-rel-def br-def)
apply (subst fmupd.abs-eq)
apply (auto simp: eq-onp-def fmap.Abs-fmap-inject
  map-drop-def map-filter-finite map-upd-def
  intro!: ext)
done

```

Technically *op-map-lookup* has the arguments in the wrong direction.

definition *fmlookup'* **where**

```
[simp]: ⟨fmlookup' A k = fmlookup k A⟩
```

lemma [*def-pat-rules*]:

```

⟨((∈#)$k$(dom-m$A)) ≡ Not$(is-None$(fmlookup'$k$A))⟩
by (simp add: fold-is-None in-fdom-alt)

```

lemma *op-map-lookup-fmlookup*:

```

⟨(op-map-lookup, fmlookup') ∈ Id → map-fmap-rel → ⟨Id⟩option-rel⟩
by (auto simp: map-fmap-rel-def br-def fmap.Abs-fmap-inverse)

```

abbreviation *hm-fmap-assn* **where**

```
⟨hm-fmap-assn K V ≡ hr-comp (hm.assn K V) map-fmap-rel⟩
```

lemmas *fmap-delete-hnr* [*sepref-fr-rules*] =
hm.delete-hnr[*FCOMP fmap-drop-set-None*]

lemmas *fmap-update-hnr* [*sepref-fr-rules*] =
hm.update-hnr[*FCOMP map-upd-fmupd*]

lemmas *fmap-lookup-hnr* [*sepref-fr-rules*] =
hm.lookup-hnr[*FCOMP op-map-lookup-fmlookup*]

lemma *fmempty-empty*:

```

⟨(uncurry0 (RETURN op-map-empty), uncurry0 (RETURN fmempty)) ∈ unit-rel →f ⟨map-fmap-rel⟩nres-rel⟩
by (auto simp: map-fmap-rel-def br-def fmempty-def frefI nres-relI)

```

lemmas [*sepref-fr-rules*] =

```
hm.empty-hnr[FCOMP fmempty-empty, unfolded op-fmap-empty-def[symmetric]]
```

abbreviation *iam-fmap-assn* **where**

```
⟨iam-fmap-assn K V ≡ hr-comp (iam.assn K V) map-fmap-rel⟩
```

lemmas *iam-fmap-delete-hnr* [*sepref-fr-rules*] =
iam.delete-hnr[*FCOMP fmap-drop-set-None*]

lemmas *iam-ffmap-update-hnr* [*sepref-fr-rules*] =
iam.update-hnr[*FCOMP map-upd-fmupd*]

lemmas *iam-ffmap-lookup-hnr* [*sepref-fr-rules*] =
iam.lookup-hnr[*FCOMP op-map-lookup-fmlookup*]

definition *op-iam-fmap-empty* **where**

⟨*op-iam-fmap-empty* = *fmempty*⟩

lemma *iam-fmempty-empty*:

⟨(*uncurry0* (*RETURN op-map-empty*), *uncurry0* (*RETURN op-iam-fmap-empty*)) ∈ *unit-rel* →_{*f*}
 ⟨*map-fmap-rel*⟩*nres-rel*⟩

by (*auto simp: map-fmap-rel-def br-def fmempty-def frefI nres-relI op-iam-fmap-empty-def*)

lemmas [*sepref-fr-rules*] =

iam.empty-hnr[*FCOMP fmempty-empty, unfolded op-iam-fmap-empty-def*[*symmetric*]]

definition *upper-bound-on-dom* **where**

⟨*upper-bound-on-dom* *A* = *SPEC*(λ*n. ∀ i ∈ #(dom-m A). i < n)*⟩

lemma [*sepref-fr-rules*]:

⟨(*Array.len*), *upper-bound-on-dom*) ∈ (*iam-fmap-assn nat-assn V*)^{*k*} →_{*a*} *nat-assn*⟩

proof –

have [*simp*]: ⟨*finite* (*dom b*) ⇒ *i* ∈ *fset* (*fmdom* (*map-of-fmap b*)) ↔ *i* ∈ *dom b*⟩ **for** *i b*

by (*subst fmdom.abs-eq*)

(*auto simp: eq-onp-def fset.Abs-fset-inverse*)

have 2: ⟨*nat-rel* = *the-pure* (*nat-assn*)⟩ **and**

3: ⟨*nat-assn* = *pure nat-rel*⟩

by *auto*

have [*simp*]: ⟨*the-pure* (λ*a c :: nat. ↑ (c = a)*) = *nat-rel*⟩

apply (*subst 2*)

apply (*subst 3*)

apply (*subst pure-def*)

apply *auto*

done

have [*simp*]: ⟨(*iam-of-list l, b*) ∈ *the-pure* (λ*a c :: nat. ↑ (c = a)*) → ⟨*the-pure V*⟩*option-rel* ⇒
b i = Some y ⇒ *i < length l*⟩ **for** *i b l y*

by (*auto dest!: fun-relD[of - - - i i] simp: option-rel-def*)

iam-of-list-def split: if-splits)

show ?*thesis*

by *sepref-to-hoare*

(*sep-auto simp: upper-bound-on-dom-def hr-comp-def iam.assn-def map-rel-def*
map-fmap-rel-def is-iam-def br-def dom-m-def)

qed

lemma *fmap-rel-nat-rel-dom-m*[*simp*]:

⟨(*A, B*) ∈ ⟨*nat-rel, R*⟩*fmap-rel* ⇒ *dom-m A* = *dom-m B*⟩

by (*subst distinct-set-mset-eq-iff*[*symmetric*])

(*auto simp: fmap-rel-alt-def distinct-mset-dom*

simp del: fmap-rel-nat-the-fmlookup)

lemma *ref-two-step'*:

⟨*A* ≤ *B* ⇒ ↓ *R A* ≤ ↓ *R B*⟩

using *ref-two-step* **by** *auto*

end

theory *PAC-Checker-Specification*

imports *PAC-Specification*

Refine-Imperative-HOL.IICF

begin

6 Checker Algorithm

In this level of refinement, we define the first level of the implementation of the checker, both with the specification as on ideals and the first version of the loop.

6.1 Specification

```
datatype status =
  is-failed: FAILED |
  is-success: SUCCESS |
  is-found: FOUND
```

```
lemma is-success-alt-def:
  ⟨is-success a ⟷ a = SUCCESS⟩
  by (cases a) auto
```

```
datatype ('a, 'b, 'lbls) pac-step =
  Add (pac-src1: 'lbls) (pac-src2: 'lbls) (new-id: 'lbls) (pac-res: 'a) |
  Mult (pac-src1: 'lbls) (pac-mult: 'a) (new-id: 'lbls) (pac-res: 'a) |
  Extension (new-id: 'lbls) (new-var: 'b) (pac-res: 'a) |
  Del (pac-src1: 'lbls)
```

```
type-synonym pac-state = ⟨(nat set × int-poly multiset)⟩
```

definition PAC-checker-specification

```
:: ⟨int-poly ⇒ int-poly multiset ⇒ (status × nat set × int-poly multiset) nres⟩
```

where

```
⟨PAC-checker-specification spec A = SPEC(λ(b, V, B).
  (¬is-failed b ⟶ restricted-ideal-to_I (⋃(vars ' set-mset A) ∪ vars spec) B ⊆ restricted-ideal-to_I
  (⋃(vars ' set-mset A) ∪ vars spec) A) ∧
  (is-found b ⟶ spec ∈ pac-ideal (set-mset A)))⟩
```

definition PAC-checker-specification-spec

```
:: ⟨int-poly ⇒ pac-state ⇒ (status × pac-state) ⇒ bool⟩
```

where

```
⟨PAC-checker-specification-spec spec = (λ(V, A) (b, B). (¬is-failed b ⟶ ⋃(vars ' set-mset A) ⊆ V) ∧
  (is-success b ⟶ PAC-Format** (V, A) B) ∧
  (is-found b ⟶ PAC-Format** (V, A) B ∧ spec ∈ pac-ideal (set-mset A)))⟩
```

abbreviation PAC-checker-specification2

```
:: ⟨int-poly ⇒ (nat set × int-poly multiset) ⇒ (status × (nat set × int-poly multiset)) nres⟩
```

where

```
⟨PAC-checker-specification2 spec A ≡ SPEC(PAC-checker-specification-spec spec A)⟩
```

definition PAC-checker-specification-step-spec

```
:: ⟨pac-state ⇒ int-poly ⇒ pac-state ⇒ (status × pac-state) ⇒ bool⟩
```

where

```
⟨PAC-checker-specification-step-spec = (λ(V0, A0) spec (V, A) (b, B).
  (is-success b ⟶
  ⋃(vars ' set-mset A0) ⊆ V0 ∧
```

$$\begin{aligned}
& \bigcup (vars \text{ ' set-mset } A) \subseteq \mathcal{V} \wedge PAC\text{-Format}^{**} (\mathcal{V}_0, A_0) (\mathcal{V}, A) \wedge PAC\text{-Format}^{**} (\mathcal{V}, A) B \wedge \\
& (is\text{-found } b \longrightarrow \\
& \quad \bigcup (vars \text{ ' set-mset } A_0) \subseteq \mathcal{V}_0 \wedge \\
& \quad \bigcup (vars \text{ ' set-mset } A) \subseteq \mathcal{V} \wedge PAC\text{-Format}^{**} (\mathcal{V}_0, A_0) (\mathcal{V}, A) \wedge PAC\text{-Format}^{**} (\mathcal{V}, A) B \wedge \\
& \quad spec \in pac\text{-ideal } (set\text{-mset } A_0))) \rangle
\end{aligned}$$

abbreviation *PAC-checker-specification-step2*

$$:: \langle pac\text{-state} \Rightarrow int\text{-poly} \Rightarrow pac\text{-state} \Rightarrow (status \times pac\text{-state}) nres \rangle$$

where

$$\langle PAC\text{-checker-specification-step2 } A_0 \text{ spec } A \equiv SPEC(PAC\text{-checker-specification-step-spec } A_0 \text{ spec } A) \rangle$$

definition *normalize-poly-spec* :: $\langle \cdot \rangle$ **where**

$$\langle normalize\text{-poly-spec } p = SPEC (\lambda r. p - r \in ideal \text{ polynomial-bool} \wedge vars \ r \subseteq vars \ p) \rangle$$

lemma *normalize-poly-spec-alt-def*:

$$\langle normalize\text{-poly-spec } p = SPEC (\lambda r. r - p \in ideal \text{ polynomial-bool} \wedge vars \ r \subseteq vars \ p) \rangle$$

unfolding *normalize-poly-spec-def*

by (*auto dest: ideal.span-neg*)

definition *mult-poly-spec* :: $\langle int \text{ mpoly} \Rightarrow int \text{ mpoly} \Rightarrow int \text{ mpoly } nres \rangle$ **where**

$$\langle mult\text{-poly-spec } p \ q = SPEC (\lambda r. p * q - r \in ideal \text{ polynomial-bool}) \rangle$$

definition *check-add* :: $\langle (nat, int \text{ mpoly}) \text{ fmap} \Rightarrow nat \text{ set} \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow int \text{ mpoly} \Rightarrow bool \text{ nres} \rangle$ **where**

$$\langle check\text{-add } A \ \mathcal{V} \ p \ q \ i \ r =$$

$$SPEC(\lambda b. b \longrightarrow p \in \# \text{ dom-m } A \wedge q \in \# \text{ dom-m } A \wedge i \notin \# \text{ dom-m } A \wedge vars \ r \subseteq \mathcal{V} \wedge \\ the \ (fmlookup \ A \ p) + the \ (fmlookup \ A \ q) - r \in ideal \text{ polynomial-bool}) \rangle$$

definition *check-mult* :: $\langle (nat, int \text{ mpoly}) \text{ fmap} \Rightarrow nat \text{ set} \Rightarrow nat \Rightarrow int \text{ mpoly} \Rightarrow nat \Rightarrow int \text{ mpoly} \Rightarrow bool \text{ nres} \rangle$ **where**

$$\langle check\text{-mult } A \ \mathcal{V} \ p \ q \ i \ r =$$

$$SPEC(\lambda b. b \longrightarrow p \in \# \text{ dom-m } A \wedge i \notin \# \text{ dom-m } A \wedge vars \ q \subseteq \mathcal{V} \wedge vars \ r \subseteq \mathcal{V} \wedge \\ the \ (fmlookup \ A \ p) * q - r \in ideal \text{ polynomial-bool}) \rangle$$

definition *check-extension* :: $\langle (nat, int \text{ mpoly}) \text{ fmap} \Rightarrow nat \text{ set} \Rightarrow nat \Rightarrow nat \Rightarrow int \text{ mpoly} \Rightarrow (bool) \text{ nres} \rangle$ **where**

$$\langle check\text{-extension } A \ \mathcal{V} \ i \ v \ p =$$

$$SPEC(\lambda b. b \longrightarrow (i \notin \# \text{ dom-m } A \wedge \\ (v \notin \mathcal{V} \wedge \\ (p + Var \ v)^2 - (p + Var \ v) \in ideal \text{ polynomial-bool} \wedge \\ vars \ (p + Var \ v) \subseteq \mathcal{V}))) \rangle$$

fun *merge-status* **where**

$$\langle merge\text{-status } (FAILED) - = FAILED \rangle |$$

$$\langle merge\text{-status } - (FAILED) = FAILED \rangle |$$

$$\langle merge\text{-status } FOUND - = FOUND \rangle |$$

$$\langle merge\text{-status } - FOUND = FOUND \rangle |$$

$$\langle merge\text{-status } - - = SUCCESS \rangle$$

type-synonym *fpac-step* = $\langle nat \text{ set} \times (nat, int\text{-poly}) \text{ fmap} \rangle$

definition *check-del* :: $\langle (nat, int \text{ mpoly}) \text{ fmap} \Rightarrow nat \Rightarrow bool \text{ nres} \rangle$ **where**

$$\langle check\text{-del } A \ p =$$

$$SPEC(\lambda b. b \longrightarrow True) \rangle$$

6.2 Algorithm

definition *PAC-checker-step*

$\langle \text{int-poly} \Rightarrow (\text{status} \times \text{fpac-step}) \Rightarrow (\text{int-poly}, \text{nat}, \text{nat}) \text{ pac-step} \Rightarrow (\text{status} \times \text{fpac-step}) \text{ nres} \rangle$

where

$\langle \text{PAC-checker-step} = (\lambda \text{spec} (\text{stat}, (\mathcal{V}, A)) \text{ st. case st of}$
 $\text{Add} \text{ - - -} \Rightarrow$
 $\text{do} \{$
 $\quad r \leftarrow \text{normalize-poly-spec} (\text{pac-res st});$
 $\quad \text{eq} \leftarrow \text{check-add } A \mathcal{V} (\text{pac-src1 st}) (\text{pac-src2 st}) (\text{new-id st}) r;$
 $\quad \text{st}' \leftarrow \text{SPEC}(\lambda \text{st}'. (\neg \text{is-failed st}' \wedge \text{is-found st}' \longrightarrow r - \text{spec} \in \text{ideal polynomial-bool}));$
 $\quad \text{if eq}$
 $\quad \text{then RETURN} (\text{merge-status stat st}',$
 $\quad \quad \mathcal{V}, \text{fmupd} (\text{new-id st}) r A)$
 $\quad \text{else RETURN} (\text{FAILED}, (\mathcal{V}, A))$
 $\quad \}$
 $| \text{Del} \text{ -} \Rightarrow$
 $\text{do} \{$
 $\quad \text{eq} \leftarrow \text{check-del } A (\text{pac-src1 st});$
 $\quad \text{if eq}$
 $\quad \text{then RETURN} (\text{stat}, (\mathcal{V}, \text{fmdrop} (\text{pac-src1 st}) A))$
 $\quad \text{else RETURN} (\text{FAILED}, (\mathcal{V}, A))$
 $\quad \}$
 $| \text{Mult} \text{ - - -} \Rightarrow$
 $\text{do} \{$
 $\quad r \leftarrow \text{normalize-poly-spec} (\text{pac-res st});$
 $\quad q \leftarrow \text{normalize-poly-spec} (\text{pac-mult st});$
 $\quad \text{eq} \leftarrow \text{check-mult } A \mathcal{V} (\text{pac-src1 st}) q (\text{new-id st}) r;$
 $\quad \text{st}' \leftarrow \text{SPEC}(\lambda \text{st}'. (\neg \text{is-failed st}' \wedge \text{is-found st}' \longrightarrow r - \text{spec} \in \text{ideal polynomial-bool}));$
 $\quad \text{if eq}$
 $\quad \text{then RETURN} (\text{merge-status stat st}',$
 $\quad \quad \mathcal{V}, \text{fmupd} (\text{new-id st}) r A)$
 $\quad \text{else RETURN} (\text{FAILED}, (\mathcal{V}, A))$
 $\quad \}$
 $| \text{Extension} \text{ - - -} \Rightarrow$
 $\text{do} \{$
 $\quad r \leftarrow \text{normalize-poly-spec} (\text{pac-res st} - \text{Var} (\text{new-var st}));$
 $\quad (\text{eq}) \leftarrow \text{check-extension } A \mathcal{V} (\text{new-id st}) (\text{new-var st}) r;$
 $\quad \text{if eq}$
 $\quad \text{then do} \{$
 $\quad \quad \text{RETURN} (\text{stat},$
 $\quad \quad \quad \text{insert} (\text{new-var st}) \mathcal{V}, \text{fmupd} (\text{new-id st}) (r) A \}$
 $\quad \text{else RETURN} (\text{FAILED}, (\mathcal{V}, A))$
 $\quad \}$
 \rangle

definition *polys-rel* $\langle ((\text{nat}, \text{int mpoly})\text{fmap} \times -) \text{ set} \rangle$ **where**

$\langle \text{polys-rel} = \{(A, B). B = (\text{ran-m } A)\} \rangle$

definition *polys-rel-full* $\langle ((\text{nat set} \times (\text{nat}, \text{int mpoly})\text{fmap}) \times -) \text{ set} \rangle$ **where**

$\langle \text{polys-rel-full} = \{((\mathcal{V}, A), (\mathcal{V}', B)). (A, B) \in \text{polys-rel} \wedge \mathcal{V} = \mathcal{V}'\} \rangle$

lemma *polys-rel-update-remove*:

$\langle x13 \notin \# \text{dom-m } A \Longrightarrow x11 \in \# \text{dom-m } A \Longrightarrow x12 \in \# \text{dom-m } A \Longrightarrow x11 \neq x12 \Longrightarrow (A, B) \in \text{polys-rel} \Longrightarrow$

```

(fmupd x13 r (fmdrop x11 (fmdrop x12 A)),
  add-mset r B - {#the (fmlookup A x11), the (fmlookup A x12)#})
  ∈ polys-rel⟩
⟨x13 ∉ #dom-m A ⇒ x11 ∈ # dom-m A ⇒ (A,B) ∈ polys-rel ⇒
  (fmupd x13 r (fmdrop x11 A), add-mset r B - {#the (fmlookup A x11)#})
  ∈ polys-rel⟩
⟨x13 ∉ #dom-m A ⇒ (A,B) ∈ polys-rel ⇒
  (fmupd x13 r A, add-mset r B) ∈ polys-rel⟩
⟨x13 ∈ #dom-m A ⇒ (A,B) ∈ polys-rel ⇒
  (fmdrop x13 A, remove1-mset (the (fmlookup A x13)) B) ∈ polys-rel⟩
using distinct-mset-dom[of A]
apply (auto simp: polys-rel-def ran-m-mapsto-upd ran-m-mapsto-upd-notin
  ran-m-fmdrop)
apply (subst ran-m-mapsto-upd-notin)
apply (auto dest: in-diffD dest!: multi-member-split simp: ran-m-fmdrop ran-m-fmdrop-If distinct-mset-remove1-All
  ran-m-def
  add-mset-eq-add-mset removeAll-notin
  split: if-splits intro!: image-mset-cong)
done

```

lemma *polys-rel-in-dom-inD*:

```

⟨(A, B) ∈ polys-rel ⇒
  x12 ∈ # dom-m A ⇒
  the (fmlookup A x12) ∈ # B⟩
by (auto simp: polys-rel-def)

```

lemma *PAC-Format-add-and-remove*:

```

⟨r - x14 ∈ More-Modules.ideal polynomial-bool ⇒
  (A, B) ∈ polys-rel ⇒
  x12 ∈ # dom-m A ⇒
  x13 ∉ # dom-m A ⇒
  vars r ⊆ V ⇒
  2 * the (fmlookup A x12) - r ∈ More-Modules.ideal polynomial-bool ⇒
  PAC-Format** (V, B) (V, remove1-mset (the (fmlookup A x12)) (add-mset r B))⟩
⟨r - x14 ∈ More-Modules.ideal polynomial-bool ⇒
  (A, B) ∈ polys-rel ⇒
  the (fmlookup A x11) + the (fmlookup A x12) - r ∈ More-Modules.ideal polynomial-bool ⇒
  x11 ∈ # dom-m A ⇒
  x12 ∈ # dom-m A ⇒
  vars r ⊆ V ⇒
  PAC-Format** (V, B) (V, add-mset r B)⟩
⟨r - x14 ∈ More-Modules.ideal polynomial-bool ⇒
  (A, B) ∈ polys-rel ⇒
  x11 ∈ # dom-m A ⇒
  x12 ∈ # dom-m A ⇒
  the (fmlookup A x11) + the (fmlookup A x12) - r ∈ More-Modules.ideal polynomial-bool ⇒
  vars r ⊆ V ⇒
  x11 ≠ x12 ⇒
  PAC-Format** (V, B)
  (V, add-mset r B - {#the (fmlookup A x11), the (fmlookup A x12)#})⟩
⟨(A, B) ∈ polys-rel ⇒
  r - x34 ∈ More-Modules.ideal polynomial-bool ⇒
  x11 ∈ # dom-m A ⇒
  the (fmlookup A x11) * x32 - r ∈ More-Modules.ideal polynomial-bool ⇒
  vars x32 ⊆ V ⇒

```

```

vars r ⊆ V ⇒
PAC-Format** (V, B) (V, add-mset r B)
⟨(A, B) ∈ polys-rel ⇒
r - x34 ∈ More-Modules.ideal polynomial-bool ⇒
x11 ∈# dom-m A ⇒
the (fmlookup A x11) * x32 - r ∈ More-Modules.ideal polynomial-bool ⇒
vars x32 ⊆ V ⇒
vars r ⊆ V ⇒
PAC-Format** (V, B) (V, remove1-mset (the (fmlookup A x11)) (add-mset r B))⟩
⟨(A, B) ∈ polys-rel ⇒
x12 ∈# dom-m A ⇒
PAC-Format** (V, B) (V, remove1-mset (the (fmlookup A x12)) B)⟩
⟨(A, B) ∈ polys-rel ⇒
(p' + Var x)2 - (p' + Var x) ∈ ideal polynomial-bool ⇒
x ∉ V ⇒
x ∉ vars(p' + Var x) ⇒
vars(p' + Var x) ⊆ V ⇒
PAC-Format** (V, B)
(insert x V, add-mset p' B)⟩
subgoal
apply (rule converse-rtranclp-into-rtranclp)
apply (rule PAC-Format.add[of ⟨the (fmlookup A x12)⟩ B ⟨the (fmlookup A x12)⟩])
apply (auto dest: polys-rel-in-dom-inD)
apply (rule converse-rtranclp-into-rtranclp)
apply (rule PAC-Format.del[of ⟨the (fmlookup A x12)⟩])
apply (auto dest: polys-rel-in-dom-inD)
done
subgoal H2
apply (rule converse-rtranclp-into-rtranclp)
apply (rule PAC-Format.add[of ⟨the (fmlookup A x11)⟩ B ⟨the (fmlookup A x12)⟩])
apply (auto dest: polys-rel-in-dom-inD)
done
subgoal
apply (rule rtranclp-trans)
apply (rule H2; assumption)
apply (rule converse-rtranclp-into-rtranclp)
apply (rule PAC-Format.del[of ⟨the (fmlookup A x12)⟩])
apply (auto dest: polys-rel-in-dom-inD)
apply (rule converse-rtranclp-into-rtranclp)
apply (rule PAC-Format.del[of ⟨the (fmlookup A x11)⟩])
apply (auto dest: polys-rel-in-dom-inD)
apply (auto simp: polys-rel-def ran-m-def add-mset-eq-add-mset dest!: multi-member-split)
done
subgoal H2
apply (rule converse-rtranclp-into-rtranclp)
apply (rule PAC-Format.mult[of ⟨the (fmlookup A x11)⟩ B ⟨x32⟩ r])
apply (auto dest: polys-rel-in-dom-inD)
done
subgoal
apply (rule rtranclp-trans)
apply (rule H2; assumption)
apply (rule converse-rtranclp-into-rtranclp)
apply (rule PAC-Format.del[of ⟨the (fmlookup A x11)⟩])
apply (auto dest: polys-rel-in-dom-inD)
done

```

subgoal

apply (*rule converse-rtranclp-into-rtranclp*)
apply (*rule PAC-Format.del*[of \langle the (fmlookup A $x12$) \rangle B])
apply (*auto dest: polys-rel-in-dom-inD*)
done

subgoal

apply (*rule converse-rtranclp-into-rtranclp*)
apply (*rule PAC-Format.extend-pos*[of $\langle p' + \text{Var } x \rangle - x$])
using *coeff-monomila-in-varsD*[of $\langle p' - \text{Var } x \rangle x$]
apply (*auto dest: polys-rel-in-dom-inD simp: vars-in-right-only vars-subst-in-left-only*)
apply (*subgoal-tac* $\langle \mathcal{V} \cup \{x' \in \text{vars } (p'). x' \notin \mathcal{V}\} = \text{insert } x \ \mathcal{V} \rangle$)
apply *simp*
using *coeff-monomila-in-varsD*[of $p' \ x$]
apply (*auto dest: vars-add-Var-subset vars-minus-Var-subset polys-rel-in-dom-inD simp: vars-subst-in-left-only-iff*)
using *vars-in-right-only vars-subst-in-left-only* **by force**
done

abbreviation *status-rel* :: \langle (*status* \times *status*) *set* \rangle **where**
 \langle *status-rel* \equiv *Id* \rangle

lemma *is-merge-status*[*simp*]:

\langle *is-failed* (*merge-status* $a \ st'$) \longleftrightarrow *is-failed* $a \ \vee$ *is-failed* st' \rangle
 \langle *is-found* (*merge-status* $a \ st'$) \longleftrightarrow \neg *is-failed* $a \ \wedge$ \neg *is-failed* $st' \ \wedge$ (*is-found* $a \ \vee$ *is-found* st') \rangle
 \langle *is-success* (*merge-status* $a \ st'$) \longleftrightarrow (*is-success* $a \ \wedge$ *is-success* st') \rangle
by (*cases a*; *cases st'*; *auto*; *fail*) $+$

lemma *status-rel-merge-status*:

\langle (*merge-status* $a \ b$, *SUCCESS*) \notin *status-rel* \longleftrightarrow
($a = \text{FAILED}$) \vee ($b = \text{FAILED}$) \vee
 $a = \text{FOUND} \ \vee$ ($b = \text{FOUND}$) \rangle
by (*cases a*; *cases b*; *auto*)

lemma *Ex-status-iff*:

\langle ($\exists a. P \ a$) \longleftrightarrow $P \ \text{SUCCESS} \ \vee$ $P \ \text{FOUND} \ \vee$ ($P \ (\text{FAILED})$) \rangle
apply *auto*
apply (*case-tac a*; *auto*)
done

lemma *is-failed-alt-def*:

\langle *is-failed* st' \longleftrightarrow \neg *is-success* $st' \ \wedge$ \neg *is-found* st' \rangle
by (*cases st'*) *auto*

lemma *merge-status-eq-iff*[*simp*]:

\langle *merge-status* $a \ \text{SUCCESS} = \text{SUCCESS} \longleftrightarrow a = \text{SUCCESS}$ \rangle
 \langle *merge-status* $a \ \text{SUCCESS} = \text{FOUND} \longleftrightarrow a = \text{FOUND}$ \rangle
 \langle *merge-status* $\text{SUCCESS} \ a = \text{SUCCESS} \longleftrightarrow a = \text{SUCCESS}$ \rangle
 \langle *merge-status* $\text{SUCCESS} \ a = \text{FOUND} \longleftrightarrow a = \text{FOUND}$ \rangle
 \langle *merge-status* $\text{SUCCESS} \ a = \text{FAILED} \longleftrightarrow a = \text{FAILED}$ \rangle
 \langle *merge-status* $a \ \text{SUCCESS} = \text{FAILED} \longleftrightarrow a = \text{FAILED}$ \rangle
 \langle *merge-status* $\text{FOUND} \ a = \text{FAILED} \longleftrightarrow a = \text{FAILED}$ \rangle
 \langle *merge-status* $a \ \text{FOUND} = \text{FAILED} \longleftrightarrow a = \text{FAILED}$ \rangle
 \langle *merge-status* $a \ \text{FOUND} = \text{SUCCESS} \longleftrightarrow \text{False}$ \rangle
 \langle *merge-status* $a \ b = \text{FOUND} \longleftrightarrow (a = \text{FOUND} \ \vee \ b = \text{FOUND}) \ \wedge$ ($a \neq \text{FAILED} \ \wedge \ b \neq \text{FAILED}$) \rangle
apply (*cases a*; *auto*; *fail*) $+$

apply (cases a; cases b; auto; fail)+
done

lemma *fmdrop-irrelevant*: $\langle x11 \notin \# \text{ dom-}m A \implies \text{fmdrop } x11 A = A \rangle$
by (*simp add: fmap-ext in-dom-m-lookup-iff*)

lemma *PAC-checker-step-PAC-checker-specification2*:

fixes *a* :: $\langle \text{status} \rangle$

assumes *AB*: $\langle (\mathcal{V}, A), (\mathcal{V}_B, B) \in \text{polys-rel-full} \rangle$ **and**

$\langle \neg \text{is-failed } a \rangle$ **and**

[*simp, intro*]: $\langle a = \text{FOUND} \implies \text{spec} \in \text{pac-ideal } (\text{set-mset } A_0) \rangle$ **and**

A₀B: $\langle \text{PAC-Format}^{**} (\mathcal{V}_0, A_0) (\mathcal{V}, B) \rangle$ **and**

spec₀: $\langle \text{vars spec} \subseteq \mathcal{V}_0 \rangle$ **and**

vars-A₀: $\langle \bigcup (\text{vars } \text{' set-mset } A_0) \subseteq \mathcal{V}_0 \rangle$

shows $\langle \text{PAC-checker-step spec } (a, (\mathcal{V}, A)) \text{ st} \leq \Downarrow (\text{status-rel} \times_r \text{polys-rel-full}) (\text{PAC-checker-specification-step2 } (\mathcal{V}_0, A_0) \text{ spec } (\mathcal{V}, B)) \rangle$

proof –

have

$\langle \mathcal{V}_B = \mathcal{V} \rangle$ **and**

[*simp, intro*]: $\langle (A, B) \in \text{polys-rel} \rangle$

using *AB*

by (*auto simp: polys-rel-full-def*)

have *H0*: $\langle 2 * \text{the } (\text{fmlookup } A \ x12) - r \in \text{More-Modules.ideal polynomial-bool} \implies$

$r \in \text{pac-ideal}$

$(\text{insert } (\text{the } (\text{fmlookup } A \ x12))$

$((\lambda x. \text{the } (\text{fmlookup } A \ x)) \text{' set-mset } Aa)) \rangle$ **for** *x12 r Aa*

by (*metis (no-types, lifting) ab-semigroup-mult-class.mult commute*

diff-in-polynomial-bool-pac-idealI

ideal.span-base pac-idealI3 set-image-mset set-mset-add-mset-insert union-single-eq-member)

then have *H0'*: $\langle \bigwedge Aa. 2 * \text{the } (\text{fmlookup } A \ x12) - r \in \text{More-Modules.ideal polynomial-bool} \implies$

$r - \text{spec} \in \text{More-Modules.ideal polynomial-bool} \implies$

$\text{spec} \in \text{pac-ideal } (\text{insert } (\text{the } (\text{fmlookup } A \ x12)) ((\lambda x. \text{the } (\text{fmlookup } A \ x)) \text{' set-mset } Aa)) \rangle$

for *r x12*

by (*metis (no-types, lifting) diff-in-polynomial-bool-pac-idealI*)

have *H1*: $\langle x12 \in \# \text{ dom-}m A \implies$

$2 * \text{the } (\text{fmlookup } A \ x12) - r \in \text{More-Modules.ideal polynomial-bool} \implies$

$r - \text{spec} \in \text{More-Modules.ideal polynomial-bool} \implies$

$\text{vars spec} \subseteq \text{vars } r \implies$

$\text{spec} \in \text{pac-ideal } (\text{set-mset } B) \rangle$ **for** *x12 r*

using $\langle (A, B) \in \text{polys-rel} \rangle$

ideal.span-add[OF ideal.span-add[OF ideal.span-neg ideal.span-neg,

of $\langle \text{the } (\text{fmlookup } A \ x12) \rangle - \langle \text{the } (\text{fmlookup } A \ x12) \rangle$ *],*

of $\langle \text{set-mset } B \cup \text{polynomial-bool} \rangle \langle 2 * \text{the } (\text{fmlookup } A \ x12) - r \rangle$ *]*

unfolding *polys-rel-def*

by (*auto dest!: multi-member-split simp: ran-m-def*

intro: H0')

have *H2'*: $\langle \text{the } (\text{fmlookup } A \ x11) + \text{the } (\text{fmlookup } A \ x12) - r \in \text{More-Modules.ideal polynomial-bool}$

\implies

$B = \text{add-mset } (\text{the } (\text{fmlookup } A \ x11)) \{ \# \text{the } (\text{fmlookup } A \ x). x \in \# Aa \# \} \implies$

$(\text{the } (\text{fmlookup } A \ x11) + \text{the } (\text{fmlookup } A \ x12) - r$

$\in \text{More-Modules.ideal}$

$(\text{insert } (\text{the } (\text{fmlookup } A \ x11))$

$((\lambda x. \text{the } (\text{fmlookup } A \ x)) \text{' set-mset } Aa \cup \text{polynomial-bool})) \implies$

$- r$

$\in \text{More-Modules.ideal}$
 $(\text{insert } (\text{the } (\text{fmlookup } A \ x11))$
 $(\lambda x. \text{the } (\text{fmlookup } A \ x)) \text{ ' set-mset } Aa \cup \text{polynomial-bool})) \implies$
 $r \in \text{pac-ideal } (\text{insert } (\text{the } (\text{fmlookup } A \ x11)) ((\lambda x. \text{the } (\text{fmlookup } A \ x)) \text{ ' set-mset } Aa))\rangle$
for $r \ x12 \ x11 \ A \ Aa$
by $(\text{metis } (\text{mono-tags, lifting}) \text{ Un-insert-left diff-diff-eq2 diff-in-polynomial-bool-pac-idealI diff-zero}$
 $\text{ideal.span-diff ideal.span-neg minus-diff-eq pac-idealI1 pac-ideal-def set-image-mset}$
 $\text{set-mset-add-mset-insert union-single-eq-member})$
have $H2: \langle x11 \in \# \text{ dom-m } A \implies$
 $x12 \in \# \text{ dom-m } A \implies$
 $\text{the } (\text{fmlookup } A \ x11) + \text{the } (\text{fmlookup } A \ x12) - r$
 $\in \text{More-Modules.ideal polynomial-bool} \implies$
 $r - \text{spec} \in \text{More-Modules.ideal polynomial-bool} \implies$
 $\text{spec} \in \text{pac-ideal } (\text{set-mset } B)\rangle$ **for** $x12 \ r \ x11$
using $\langle (A,B) \in \text{polys-rel} \rangle$
 $\text{ideal.span-add}[OF \ \text{ideal.span-add}[OF \ \text{ideal.span-neg ideal.span-neg,}$
 $\text{of } \langle \text{the } (\text{fmlookup } A \ x11) \rangle - \langle \text{the } (\text{fmlookup } A \ x12) \rangle],$
 $\text{of } \langle \text{set-mset } B \cup \text{polynomial-bool} \rangle \langle \text{the } (\text{fmlookup } A \ x11) + \text{the } (\text{fmlookup } A \ x12) - r \rangle]$
unfolding polys-rel-def
by $(\text{subgoal-tac } \langle r \in \text{pac-ideal } (\text{set-mset } B) \rangle)$
 $(\text{auto dest!: multi-member-split simp: ran-m-def ideal.span-base}$
 $\text{intro: diff-in-polynomial-bool-pac-idealI simp: } H2')$

have $H3': \langle \text{the } (\text{fmlookup } A \ x12) * q - r \in \text{More-Modules.ideal polynomial-bool} \implies$
 $r - \text{spec} \in \text{More-Modules.ideal polynomial-bool} \implies$
 $r \in \text{pac-ideal } (\text{insert } (\text{the } (\text{fmlookup } A \ x12)) ((\lambda x. \text{the } (\text{fmlookup } A \ x)) \text{ ' set-mset } Aa))\rangle$
for $Aa \ x12 \ r \ q$
by $(\text{metis } (\text{no-types, lifting}) \text{ ab-semigroup-mult-class.mult commute diff-in-polynomial-bool-pac-idealI}$
 $\text{ideal.span-base pac-idealI3 set-image-mset set-mset-add-mset-insert union-single-eq-member})$

have $H3: \langle x12 \in \# \text{ dom-m } A \implies$
 $\text{the } (\text{fmlookup } A \ x12) * q - r \in \text{More-Modules.ideal polynomial-bool} \implies$
 $r - \text{spec} \in \text{More-Modules.ideal polynomial-bool} \implies$
 $\text{spec} \in \text{pac-ideal } (\text{set-mset } B)\rangle$ **for** $x12 \ r \ q$
using $\langle (A,B) \in \text{polys-rel} \rangle$
 $\text{ideal.span-add}[OF \ \text{ideal.span-add}[OF \ \text{ideal.span-neg ideal.span-neg,}$
 $\text{of } \langle \text{the } (\text{fmlookup } A \ x12) \rangle - \langle \text{the } (\text{fmlookup } A \ x12) \rangle],$
 $\text{of } \langle \text{set-mset } B \cup \text{polynomial-bool} \rangle \langle 2 * \text{the } (\text{fmlookup } A \ x12) - r \rangle]$
unfolding polys-rel-def
by $(\text{subgoal-tac } \langle r \in \text{pac-ideal } (\text{set-mset } B) \rangle)$
 $(\text{auto dest!: multi-member-split simp: ran-m-def } H3')$
 $\text{intro: diff-in-polynomial-bool-pac-idealI}$

have $[\text{intro}]: \langle \text{spec} \in \text{pac-ideal } (\text{set-mset } B) \implies \text{spec} \in \text{pac-ideal } (\text{set-mset } A_0)\rangle$ **and**
 $\text{vars-B: } \langle \bigcup (\text{vars } \text{ ' set-mset } B) \subseteq \mathcal{V} \rangle$ **and**
 $\text{vars-B: } \langle \bigcup (\text{vars } \text{ ' set-mset } (\text{ran-m } A)) \subseteq \mathcal{V} \rangle$
using $\text{rtranclp-PAC-Format-subset-ideal}[OF \ A_0 B \ \text{vars-A}_0] \text{spec}_0 \langle (A, B) \in \text{polys-rel} \rangle [\text{unfolded}$
 $\text{polys-rel-def, simplified}]$
by $(\text{smt in-mono mem-Collect-eq restricted-ideal-to-def})+$

have $\text{eq-successI: } \langle st' \neq \text{FAILED} \implies$
 $st' \neq \text{FOUND} \implies st' = \text{SUCCESS} \rangle$ **for** st'
by $(\text{cases } st') \text{ auto}$
have $\text{vars-diff-inv: } \langle \text{vars } (\text{Var } x2 - r) = \text{vars } (r - \text{Var } x2 :: \text{int mpoly}) \rangle$ **for** $x2 \ r$
using $\text{vars-uminus}[of \ \langle \text{Var } x2 - r \rangle]$


```

by (auto simp del: vars-uminus)
have vars-add-inv: ⟨vars (Var x2 + r) = vars (r + Var x2 :: int mpoly)⟩ for x2 r
  unfolding add.commute[of ⟨Var x2⟩ r] ..

have [iff]: ⟨a ≠ FAILED⟩ and
  [intro]: ⟨a ≠ SUCCESS ⇒ a = FOUND⟩ and
  [simp]: ⟨merge-status a FOUND = FOUND⟩
  using assms(2) by (cases a; auto)+
note [[goals-limit=1]]
show ?thesis
  unfolding PAC-checker-step-def PAC-checker-specification-step-spec-def
    normalize-poly-spec-alt-def check-mult-def check-add-def
    check-extension-def polys-rel-full-def
  apply (cases st)
  apply clarsimp-all
  subgoal for x11 x12 x13 x14
    apply (refine-vcg lhs-step-If)
    subgoal for r eqa st'
      using assms vars-B apply –
      apply (rule RETURN-SPEC-refine)
      apply (rule-tac x = ⟨(merge-status a st', V, add-mset r B)⟩ in exI)
      by (auto simp: polys-rel-update-remove ran-m-mapsto-upd-notin
        intro: PAC-Format-add-and-remove H2 dest: rtranclp-PAC-Format-subset-ideal)
    subgoal
      by (rule RETURN-SPEC-refine)
      (auto simp: Ex-status-iff dest: rtranclp-PAC-Format-subset-ideal)
    done
  subgoal for x11 x12 x13 x14
    apply (refine-vcg lhs-step-If)
    subgoal for r q eqa st'
      using assms vars-B apply –
      apply (rule RETURN-SPEC-refine)
      apply (rule-tac x = ⟨(merge-status a st', V, add-mset r B)⟩ in exI)
      by (auto intro: polys-rel-update-remove intro: PAC-Format-add-and-remove(3–) H3
        dest: rtranclp-PAC-Format-subset-ideal)
    subgoal
      by (rule RETURN-SPEC-refine)
      (auto simp: Ex-status-iff)
    done
  subgoal for x31 x32 x34
    apply (refine-vcg lhs-step-If)
    subgoal for r x
      using assms vars-B apply –
      apply (rule RETURN-SPEC-refine)
      apply (rule-tac x = ⟨a, insert x32 V, add-mset r B)⟩ in exI)
      apply (auto simp: intro!: polys-rel-update-remove PAC-Format-add-and-remove(5–)
        dest: rtranclp-PAC-Format-subset-ideal)
    done
  subgoal
    by (rule RETURN-SPEC-refine)
    (auto simp: Ex-status-iff)
  done
  subgoal for x11
    unfolding check-del-def
    apply (refine-vcg lhs-step-If)

```

```

subgoal for eq
  using assms vars-B apply –
  apply (rule RETURN-SPEC-refine)
  apply (cases  $\langle x11 \in \# \text{ dom-}m \ A \rangle$ )
  subgoal
    apply (rule-tac  $x = \langle (a, \mathcal{V}, \text{remove1-mset } (\text{the } (\text{fmlookup } A \ x11)) \ B) \rangle$  in exI)
    apply (auto simp: polys-rel-update-remove PAC-Format-add-and-remove
      is-failed-def is-success-def is-found-def
      dest!: eq-successI
      split: if-splits
      dest: rtranclp-PAC-Format-subset-ideal
      intro: PAC-Format-add-and-remove H3)
    done
  subgoal
    apply (rule-tac  $x = \langle (a, \mathcal{V}, B) \rangle$  in exI)
    apply (auto simp: fmdrop-irrelevant
      is-failed-def is-success-def is-found-def
      dest!: eq-successI
      split: if-splits
      dest: rtranclp-PAC-Format-subset-ideal
      intro: PAC-Format-add-and-remove)
    done
  done
done
subgoal
  by (rule RETURN-SPEC-refine)
  (auto simp: Ex-status-iff)
done
done
qed

```

definition *PAC-checker*

$:: \langle \text{int-poly} \Rightarrow \text{fpac-step} \Rightarrow \text{status} \Rightarrow (\text{int-poly}, \text{nat}, \text{nat}) \text{ pac-step list} \Rightarrow$
 $(\text{status} \times \text{fpac-step}) \text{ nres} \rangle$

where

```

 $\langle \text{PAC-checker spec } A \ b \ st = \text{do} \{$ 
   $(S, -) \leftarrow \text{WHILE}_T$ 
   $(\lambda((b :: \text{status}, A :: \text{fpac-step}), st). \neg \text{is-failed } b \wedge st \neq [])$ 
   $(\lambda((bA), st). \text{do} \{$ 
     $\text{ASSERT}(st \neq [])$ ;
     $S \leftarrow \text{PAC-checker-step spec } (bA) \ (\text{hd } st)$ ;
     $\text{RETURN } (S, \text{tl } st)$ 
   $\})$ 
   $((b, A), st)$ ;
 $\text{RETURN } S$ 
 $\} \rangle$ 

```

lemma *PAC-checker-specification-spec-trans:*

$\langle \text{PAC-checker-specification-spec spec } A \ (st, x2) \Longrightarrow$
 $\text{PAC-checker-specification-step-spec } A \ \text{spec } x2 \ (st', x1a) \Longrightarrow$
 $\text{PAC-checker-specification-spec spec } A \ (st', x1a) \rangle$

unfolding *PAC-checker-specification-spec-def*

PAC-checker-specification-step-spec-def

apply *auto*

using *is-failed-alt-def* **apply** *blast+*
done

lemma *RES-SPEC-eq*:
 $\langle RES \ \Phi = SPEC(\lambda P. P \in \Phi) \rangle$
by *auto*

lemma *is-failed-is-success-completeD*:
 $\langle \neg \text{is-failed } x \implies \neg \text{is-success } x \implies \text{is-found } x \rangle$
by (*cases x*) *auto*

lemma *PAC-checker-PAC-checker-specification2*:
 $\langle (A, B) \in \text{polys-rel-full} \implies$
 $\neg \text{is-failed } a \implies$
 $(a = FOUND \implies \text{spec} \in \text{pac-ideal } (\text{set-mset } (\text{snd } B))) \implies$
 $\bigcup (\text{vars } ' \text{set-mset } (\text{ran-m } (\text{snd } A))) \subseteq \text{fst } B \implies$
 $\text{vars } \text{spec} \subseteq \text{fst } B \implies$
 $\text{PAC-checker } \text{spec } A \ a \ st \leq \Downarrow (\text{status-rel } \times_r \ \text{polys-rel-full}) (\text{PAC-checker-specification2 } \text{spec } B) \rangle$
unfolding *PAC-checker-def conc-fun-RES*
apply (*subst RES-SPEC-eq*)
apply (*refine-vcg WHILET-rule*[**where**
 $I = \langle \lambda((bB), st). \ bB \in (\text{status-rel } \times_r \ \text{polys-rel-full})^{-1} \ \langle \langle$
 $\text{Collect } (\text{PAC-checker-specification-spec } \text{spec } B) \rangle \rangle$
and $R = \langle \text{measure } (\lambda(-, st). \ \text{Suc } (\text{length } st)) \rangle$]
subgoal **by** *auto*
subgoal **apply** (*auto simp: PAC-checker-specification-spec-def*)
apply (*cases B; cases A*)
apply (*auto simp: polys-rel-def polys-rel-full-def Image-iff*)
done
subgoal **by** *auto*
subgoal
apply *auto*
apply (*rule*
 $\text{PAC-checker-step-PAC-checker-specification2}[\text{of } \text{----- } \langle \text{fst } B \rangle, \text{ THEN } \text{order-trans}]$)
apply *assumption*
apply *assumption*
apply (*auto intro: PAC-checker-specification-spec-trans simp: conc-fun-RES*)
apply (*auto simp: PAC-checker-specification-spec-def polys-rel-full-def polys-rel-def*
 $\text{dest: PAC-Format-subset-ideal}$
 $\text{dest: is-failed-is-success-completeD; fail}$)
by (*auto simp: Image-iff intro: PAC-checker-specification-spec-trans*
 $\text{simp: polys-rel-def polys-rel-full-def}$)
subgoal
by *auto*
done

definition *remap-polys-polynomial-bool* :: $\langle \text{int } \text{mpoly} \Rightarrow \text{nat } \text{set} \Rightarrow (\text{nat}, \text{int-poly}) \text{ fmap} \Rightarrow (\text{status} \times \text{fpac-step}) \text{ nres} \rangle$ **where**
 $\langle \text{remap-polys-polynomial-bool } \text{spec} = (\lambda \mathcal{V} \ A.$
 $SPEC(\lambda(st, \mathcal{V}', A'). (\neg \text{is-failed } st \longrightarrow$
 $\text{dom-m } A = \text{dom-m } A' \wedge$
 $(\forall i \in \# \text{dom-m } A. \ \text{the } (\text{fmlookup } A \ i) - \text{the } (\text{fmlookup } A' \ i) \in \text{ideal } \text{polynomial-bool}) \wedge$
 $\bigcup (\text{vars } ' \text{set-mset } (\text{ran-m } A)) \subseteq \mathcal{V}' \wedge$
 $\bigcup (\text{vars } ' \text{set-mset } (\text{ran-m } A')) \subseteq \mathcal{V}') \wedge$
 $(st = FOUND \longrightarrow \text{spec} \in \# \text{ran-m } A')) \rangle$

definition *remap-polys-change-all* :: $\langle \text{int mpoly} \Rightarrow \text{nat set} \Rightarrow (\text{nat}, \text{int-poly}) \text{ fmap} \Rightarrow (\text{status} \times \text{fpac-step}) \text{ nres} \rangle$ **where**

$\langle \text{remap-polys-change-all spec} = (\lambda \mathcal{V} A. \text{SPEC} (\lambda (st, \mathcal{V}', A').$

$(\neg \text{is-failed } st \longrightarrow$
 $\text{pac-ideal } (\text{set-mset } (\text{ran-m } A)) = \text{pac-ideal } (\text{set-mset } (\text{ran-m } A')) \wedge$
 $\bigcup (\text{vars } ' \text{set-mset } (\text{ran-m } A)) \subseteq \mathcal{V}' \wedge$
 $\bigcup (\text{vars } ' \text{set-mset } (\text{ran-m } A')) \subseteq \mathcal{V}' \wedge$
 $(st = \text{FOUND} \longrightarrow \text{spec} \in \# \text{ran-m } A')) \rangle$

lemma *fmap-eq-dom-iff*:

$\langle A = A' \longleftrightarrow \text{dom-m } A = \text{dom-m } A' \wedge (\forall i \in \# \text{dom-m } A. \text{the } (\text{fmlookup } A \ i) = \text{the } (\text{fmlookup } A' \ i)) \rangle$
by (*metis fmap-ext in-dom-m-lookup-iff option.expand*)

lemma *ideal-remap-incl*:

$\langle \text{finite } A' \Longrightarrow (\forall a' \in A'. \exists a \in A. a - a' \in B) \Longrightarrow \text{ideal } (A' \cup B) \subseteq \text{ideal } (A \cup B) \rangle$

apply (*induction A' rule: finite-induct*)

apply (*auto intro: ideal.span-mono*)

using *ideal.span-mono sup-ge2* **apply** *blast*

proof –

fix $x :: 'a$ **and** $F :: 'a \text{ set}$ **and** $xa :: 'a$ **and** $a :: 'a$

assume $a1: a \in A$

assume $a2: a - x \in B$

assume $a3: xa \in \text{More-Modules.ideal } (\text{insert } x \ (F \cup B))$

assume $a4: \text{More-Modules.ideal } (F \cup B) \subseteq \text{More-Modules.ideal } (A \cup B)$

have $x \in \text{More-Modules.ideal } (A \cup B)$

using $a2 \ a1$ **by** (*metis (no-types, lifting) Un-upper1 Un-upper2 add-diff-cancel-left' diff-add-cancel ideal.module-axioms ideal.span-diff in-mono module.span-superset*)

then show $xa \in \text{More-Modules.ideal } (A \cup B)$

using $a4 \ a3$ *ideal.span-insert-subset* **by** *blast*

qed

lemma *pac-ideal-remap-eq*:

$\langle \text{dom-m } b = \text{dom-m } ba \Longrightarrow$

$\forall i \in \# \text{dom-m } ba.$

$\text{the } (\text{fmlookup } b \ i) - \text{the } (\text{fmlookup } ba \ i)$

$\in \text{More-Modules.ideal polynomial-bool} \Longrightarrow$

$\text{pac-ideal } ((\lambda x. \text{the } (\text{fmlookup } b \ x)) ' \text{set-mset } (\text{dom-m } ba)) = \text{pac-ideal } ((\lambda x. \text{the } (\text{fmlookup } ba \ x)) '$

$\text{set-mset } (\text{dom-m } ba)) \rangle$

unfolding *pac-ideal-alt-def*

apply *standard*

subgoal

apply (*rule ideal-remap-incl*)

apply (*auto dest!: multi-member-split*

dest: ideal.span-neg)

apply (*drule ideal.span-neg*)

apply *auto*

done

subgoal

by (*rule ideal-remap-incl*)

(*auto dest!: multi-member-split*)

done

lemma *remap-polys-polynomial-bool-remap-polys-change-all*:

$\langle \text{remap-polys-polynomial-bool spec } \mathcal{V} \ A \leq \text{remap-polys-change-all spec } \mathcal{V} \ A \rangle$

unfolding *remap-polys-polynomial-bool-def remap-polys-change-all-def*
apply (*simp add: ideal.span-zero fmap-eq-dom-iff ideal.span-eq*)
apply (*auto dest: multi-member-split simp: ran-m-def ideal.span-base pac-ideal-remap-eq*
add-mset-eq-add-mset
eq-commute[of ‹add-mset - -› ‹dom-m (A :: (nat, int mpoly)fmap)› for A])
done

definition *remap-polys* :: ‹*int mpoly* \Rightarrow *nat set* \Rightarrow (*nat, int-poly*) *fmap* \Rightarrow (*status* \times *fpac-step*) *nres*›
where

‹*remap-polys spec* = ($\lambda \mathcal{V} A$. *do*{
dom \leftarrow *SPEC*(λdom . *set-mset* (*dom-m* *A*) \subseteq *dom* \wedge *finite* *dom*);

failed \leftarrow *SPEC*($\lambda :: bool$. *True*);
if *failed*
then *do* {
RETURN (*FAILED*, \mathcal{V} , *fmempty*)
} }
else *do* {
(*b*, *N*) \leftarrow *FOREACH* *dom*
(λi (*b*, \mathcal{V} , *A'*).
if $i \in \#$ *dom-m* *A*
then *do* {
p \leftarrow *SPEC*(λp . *the* (*fmlookup* *A* *i*) – $p \in$ *ideal polynomial-bool* \wedge *vars* $p \subseteq$ *vars* (*the* (*fmlookup* *A* *i*)));
eq \leftarrow *SPEC*(λeq . $eq \longrightarrow p = spec$);
 $\mathcal{V} \leftarrow$ *SPEC*($\lambda \mathcal{V}'$. $\mathcal{V} \cup vars$ (*the* (*fmlookup* *A* *i*)) $\subseteq \mathcal{V}'$);
RETURN($b \vee eq$, \mathcal{V} , *fmupd* *i* *p* *A'*)
} *else* *RETURN* (*b*, \mathcal{V} , *A'*)
(*False*, \mathcal{V} , *fmempty*);
RETURN (*if* *b* *then* *FOUND* *else* *SUCCESS*, *N*)
} })›

lemma *remap-polys-spec*:

‹*remap-polys spec* \mathcal{V} *A* \leq *remap-polys-polynomial-bool spec* \mathcal{V} *A*›
unfolding *remap-polys-def remap-polys-polynomial-bool-def*
apply (*refine-vcg FOREACH-rule*[**where**
I = ‹ λdom (*b*, \mathcal{V} , *A'*).
set-mset (*dom-m* *A'*) = *set-mset* (*dom-m* *A*) – *dom* \wedge
($\forall i \in set-mset$ (*dom-m* *A*) – *dom*. *the* (*fmlookup* *A* *i*) – *the* (*fmlookup* *A'* *i*) \in *ideal polynomial-bool*)
 \wedge
 \bigcup (*vars* ‹*set-mset* (*ran-m* (*fmrestrict-set* (*set-mset* (*dom-m* *A'*)) *A*))›) $\subseteq \mathcal{V} \wedge$
 \bigcup (*vars* ‹*set-mset* (*ran-m* *A'*)›) $\subseteq \mathcal{V} \wedge$
($b \longrightarrow spec \in \#$ *ran-m* *A'*)›)])

subgoal *by auto*

subgoal *by auto*

subgoal *by auto*

subgoal *by auto*

subgoal *by auto*

subgoal *by auto*

subgoal *by auto*

subgoal

by auto

subgoal *by auto*

```

subgoal using ideal.span-add by auto
subgoal by auto
subgoal by auto
subgoal by clarsimp auto
subgoal
  supply[[goals-limit=1]]
  by (auto simp add: ran-m-mapsto-upd-notin dom-m-fmrestrict-set' subset-eq)
subgoal
  supply[[goals-limit=1]]
  by (auto simp add: ran-m-mapsto-upd-notin dom-m-fmrestrict-set' subset-eq)
subgoal
  by (auto simp: ran-m-mapsto-upd-notin)
subgoal
  by auto
subgoal
  by auto
subgoal
  by (auto simp add: ran-m-mapsto-upd-notin dom-m-fmrestrict-set' subset-eq)
subgoal
  by (auto simp add: ran-m-mapsto-upd-notin dom-m-fmrestrict-set' subset-eq)
subgoal
  by auto
subgoal
  by (auto simp: distinct-set-mset-eq-iff[symmetric] distinct-mset-dom)
subgoal
  by (auto simp: distinct-set-mset-eq-iff[symmetric] distinct-mset-dom)
subgoal
  by (auto simp add: ran-m-mapsto-upd-notin dom-m-fmrestrict-set' subset-eq
    fmlookup-restrict-set-id')
subgoal
  by (auto simp add: ran-m-mapsto-upd-notin dom-m-fmrestrict-set' subset-eq)
subgoal
  by (auto simp add: ran-m-mapsto-upd-notin dom-m-fmrestrict-set' subset-eq
    fmlookup-restrict-set-id')
done

```

6.3 Full Checker

definition *full-checker*

$:: \langle \text{int-poly} \Rightarrow (\text{nat}, \text{int-poly}) \text{ fmap} \Rightarrow (\text{int-poly}, \text{nat}, \text{nat}) \text{ pac-step list} \Rightarrow (\text{status} \times _) \text{ nres} \rangle$

where

```

⟨full-checker spec0 A pac = do {
  spec ← normalize-poly-spec spec0;
  (st, V, A) ← remap-polys-change-all spec {} A;
  if is-failed st then
    RETURN (st, V, A)
  else do {
    V ← SPEC( $\lambda \mathcal{V}'$ .  $\mathcal{V} \cup \text{vars spec0} \subseteq \mathcal{V}'$ );
    PAC-checker spec (V, A) st pac
  }
}

```

}⟩

lemma *restricted-ideal-to-mono*:

$\langle \text{restricted-ideal-to}_I \mathcal{V} I \subseteq \text{restricted-ideal-to}_I \mathcal{V}' J \implies$

$\mathcal{U} \subseteq \mathcal{V} \implies$

$\text{restricted-ideal-to}_I \mathcal{U} I \subseteq \text{restricted-ideal-to}_I \mathcal{U} J \rangle$

by (auto simp: restricted-ideal-to-def)

lemma *pac-ideal-idemp*: $\langle \text{pac-ideal} (\text{pac-ideal } A) = \text{pac-ideal } A \rangle$

by (metis dual-order.antisym ideal.span-subset-spanI ideal.span-superset le-sup-iff pac-ideal-def)

lemma *full-checker-spec*:

assumes $\langle (A, A') \in \text{polys-rel} \rangle$

shows

$\langle \text{full-checker spec } A \text{ pac} \leq \Downarrow \{((st, G), (st', G')). (st, st') \in \text{status-rel} \wedge$
 $(st \neq \text{FAILED} \longrightarrow (G, G') \in \text{polys-rel-full})\}$
 $(\text{PAC-checker-specification spec } (A')) \rangle$

proof –

have *H*: $\langle \text{set-mset } b \subseteq \text{pac-ideal} (\text{set-mset} (\text{ran-m } A)) \implies$

$x \in \text{pac-ideal} (\text{set-mset } b) \implies x \in \text{pac-ideal} (\text{set-mset } A') \rangle$ **for** *b x*

using *assms* **apply** –

by (drule *pac-ideal-mono*) (auto simp: *polys-rel-def pac-ideal-idemp*)

have *1*: $\langle x \in \{(st, \mathcal{V}', A')\}$.

$(\neg \text{is-failed } st \longrightarrow \text{pac-ideal} (\text{set-mset} (\text{ran-m } x2)) =$
 $\text{pac-ideal} (\text{set-mset} (\text{ran-m } A')) \wedge$
 $\bigcup (\text{vars } ' \text{set-mset} (\text{ran-m } ABC)) \subseteq \mathcal{V}' \wedge$
 $\bigcup (\text{vars } ' \text{set-mset} (\text{ran-m } A')) \subseteq \mathcal{V}' \wedge$
 $(st = \text{FOUND} \longrightarrow \text{spec} \in \# \text{ran-m } A') \rangle \implies$

$x = (st, x') \implies x' = (\mathcal{V}, Aa) \implies ((\mathcal{V}', Aa), \mathcal{V}', \text{ran-m } Aa) \in \text{polys-rel-full} \rangle$ **for** *Aa spec x2 st x*
 $\mathcal{V}' \mathcal{V} x' ABC$

by (auto simp: *polys-rel-def polys-rel-full-def*)

have *H1*: $\langle \bigwedge a aa b xa x x1a x1 x2 \text{spec}.$

$\text{vars spec} \subseteq x1b \implies$

$\bigcup (\text{vars } ' \text{set-mset} (\text{ran-m } A)) \subseteq x1b \implies$

$\bigcup (\text{vars } ' \text{set-mset} (\text{ran-m } x2a)) \subseteq x1b \implies$

$\text{restricted-ideal-to}_I x1b b \subseteq \text{restricted-ideal-to}_I x1b (\text{ran-m } x2a) \implies$

$xa \in \text{restricted-ideal-to}_I (\bigcup (\text{vars } ' \text{set-mset} (\text{ran-m } A)) \cup \text{vars spec}) b \implies$

$xa \in \text{restricted-ideal-to}_I (\bigcup (\text{vars } ' \text{set-mset} (\text{ran-m } A)) \cup \text{vars spec}) (\text{ran-m } x2a) \rangle$

for *x1b b xa x2a*

by (drule *restricted-ideal-to-mono*[of - - - $\langle \bigcup (\text{vars } ' \text{set-mset} (\text{ran-m } A)) \cup \text{vars spec} \rangle$])

auto

have *H2*: $\langle \bigwedge a aa b \text{spec} x2 x1a x1b x2a.$

$\text{spec} - \text{spec} \in \text{More-Modules.ideal polynomial-bool} \implies$

$\text{vars spec} \subseteq x1b \implies$

$\bigcup (\text{vars } ' \text{set-mset} (\text{ran-m } A)) \subseteq x1b \implies$

$\bigcup (\text{vars } ' \text{set-mset} (\text{ran-m } x2a)) \subseteq x1b \implies$

$\text{spec} \in \text{pac-ideal} (\text{set-mset} (\text{ran-m } x2a)) \implies$

$\text{restricted-ideal-to}_I x1b b \subseteq \text{restricted-ideal-to}_I x1b (\text{ran-m } x2a) \implies$

$\text{spec} \in \text{pac-ideal} (\text{set-mset} (\text{ran-m } x2a)) \rangle$

by (metis (no-types, lifting) *group-eq-aux ideal.span-add ideal.span-base in-mono*)

pac-ideal-alt-def sup.cobounded2)

show *?thesis*

supply[[*goals-limit=1*]]

unfolding *full-checker-def normalize-poly-spec-def*

PAC-checker-specification-def remap-polys-change-all-def

apply (*refine-vcg PAC-checker-PAC-checker-specification2*[*THEN order-trans, of - -*]

lhs-step-If)

subgoal **by** (auto simp: *is-failed-def RETURN-RES-refine-iff*)

apply (*rule 1; assumption*)

subgoal

```

    using fmap-ext assms by (auto simp: polys-rel-def ran-m-def)
  subgoal
    by auto
  subgoal
    by auto
  subgoal for spec a x1 x2 x x1a x2a x1b
    apply (rule ref-two-step[OF conc-fun-R-mono])
    apply auto[]
    using assms
    by (auto simp add: PAC-checker-specification-spec-def conc-fun-RES polys-rel-def H1 H2
        polys-rel-full-def
        dest!: rtranclp-PAC-Format-subset-ideal dest: is-failed-is-success-completeD)
  done
qed

```

```

lemma full-checker-spec':
  shows
    ⟨(uncurry2 full-checker, uncurry2 (λspec A -. PAC-checker-specification spec A)) ∈
      (Id ×r polys-rel) ×r Id →f ⟨{(st, G), (st', G')}. (st, st') ∈ status-rel ∧
      (st ≠ FAILED → (G, G') ∈ polys-rel-full)}⟩nres-rel⟩
  using full-checker-spec
  by (auto intro!: frefl nres-relI)

```

```

end
theory PAC-Polynomials
  imports PAC-Specification Finite-Map-Multiset
begin

```

7 Polynomials of strings

Isabelle's definition of polynomials only work with variables of type *nat*. Therefore, we introduce a version that uses strings by using an injective function that converts a string to a natural number. It exists because strings are countable. Remark that the whole development is independent of the function.

7.1 Polynomials and Variables

```

lemma poly-embed-EX:
  ⟨∃φ. bij (φ :: string ⇒ nat)⟩
  by (rule countableE-infinite[of ⟨UNIV :: string set⟩])
  (auto intro!: infinite-UNIV-listI)

```

Using a multiset instead of a list has some advantage from an abstract point of view. First, we can have monomials that appear several times and the coefficient can also be zero. Basically, we can represent un-normalised polynomials, which is very useful to talk about intermediate states in our program.

```

type-synonym term-poly = ⟨string multiset⟩
type-synonym mset-polynomial =
  ⟨(term-poly * int) multiset⟩

```

```

definition normalized-poly :: ⟨mset-polynomial ⇒ bool⟩ where
  ⟨normalized-poly p ⟷

```


$distinct\text{-}mset (fst \text{'\# } p) \wedge$
 $0 \notin \# snd \text{'\# } p\rangle$

lemma *normalized-poly-simps*[simp]:

$\langle normalized\text{-}poly \ \{\#\} \rangle$
 $\langle normalized\text{-}poly (add\text{-}mset \ t \ p) \longleftrightarrow snd \ t \neq 0 \wedge$
 $\quad fst \ t \notin \# \text{'\# } p \wedge normalized\text{-}poly \ p \rangle$
by (auto simp: *normalized-poly-def*)

lemma *normalized-poly-mono*:

$\langle normalized\text{-}poly \ B \implies A \subseteq \# B \implies normalized\text{-}poly \ A \rangle$
unfolding *normalized-poly-def*
by (auto intro: *distinct-mset-mono image-mset-subseteq-mono*)

definition *mult-poly-by-monom* :: $\langle term\text{-}poly * int \Rightarrow mset\text{-}polynomial \Rightarrow mset\text{-}polynomial \rangle$ **where**
 $\langle mult\text{-}poly\text{-}by\text{-}monom = (\lambda ys \ q. image\text{-}mset (\lambda xs. (fst \ xs + fst \ ys, snd \ ys * snd \ xs)) \ q) \rangle$

definition *mult-poly-raw* :: $\langle mset\text{-}polynomial \Rightarrow mset\text{-}polynomial \Rightarrow mset\text{-}polynomial \rangle$ **where**

$\langle mult\text{-}poly\text{-}raw \ p \ q =$
 $\quad (sum\text{-}mset ((\lambda y. mult\text{-}poly\text{-}by\text{-}monom \ y \ q) \ \text{'\# } p)) \rangle$

definition *remove-powers* :: $\langle mset\text{-}polynomial \Rightarrow mset\text{-}polynomial \rangle$ **where**

$\langle remove\text{-}powers \ xs = image\text{-}mset (apfst \ remdups\text{-}mset) \ xs \rangle$

definition *all-vars-mset* :: $\langle mset\text{-}polynomial \Rightarrow string \ multiset \rangle$ **where**

$\langle all\text{-}vars\text{-}mset \ p = \sum \# (fst \ \text{'\# } p) \rangle$

abbreviation *all-vars* :: $\langle mset\text{-}polynomial \Rightarrow string \ set \rangle$ **where**

$\langle all\text{-}vars \ p \equiv set\text{-}mset (all\text{-}vars\text{-}mset \ p) \rangle$

definition *add-to-coefficient* :: $\langle - \Rightarrow mset\text{-}polynomial \Rightarrow mset\text{-}polynomial \rangle$ **where**

$\langle add\text{-}to\text{-}coefficient = (\lambda (a, n) \ b. \{\#\ (a', -) \in \# \ b. \ a' \neq a\#\} +$
 $\quad (if \ n + sum\text{-}mset (snd \ \text{'\# } \{\#\ (a', -) \in \# \ b. \ a' = a\#\}) = 0 \ then \ \{\#\}$
 $\quad \quad \text{else } \{\#\ (a, n + sum\text{-}mset (snd \ \text{'\# } \{\#\ (a', -) \in \# \ b. \ a' = a\#\}))\#\}) \rangle$

definition *normalize-poly* :: $\langle mset\text{-}polynomial \Rightarrow mset\text{-}polynomial \rangle$ **where**

$\langle normalize\text{-}poly \ p = fold\text{-}mset \ add\text{-}to\text{-}coefficient \ \{\#\} \ p \rangle$

lemma *add-to-coefficient-simps*:

$\langle n + sum\text{-}mset (snd \ \text{'\# } \{\#\ (a', -) \in \# \ b. \ a' = a\#\}) \neq 0 \implies$
 $\quad add\text{-}to\text{-}coefficient \ (a, n) \ b = \{\#\ (a', -) \in \# \ b. \ a' \neq a\#\} +$
 $\quad \quad \{\#\ (a, n + sum\text{-}mset (snd \ \text{'\# } \{\#\ (a', -) \in \# \ b. \ a' = a\#\}))\#\} \rangle$
 $\langle n + sum\text{-}mset (snd \ \text{'\# } \{\#\ (a', -) \in \# \ b. \ a' = a\#\}) = 0 \implies$
 $\quad add\text{-}to\text{-}coefficient \ (a, n) \ b = \{\#\ (a', -) \in \# \ b. \ a' \neq a\#\} \rangle$ **and**
add-to-coefficient-simps-If:
 $\langle add\text{-}to\text{-}coefficient \ (a, n) \ b = \{\#\ (a', -) \in \# \ b. \ a' \neq a\#\} +$
 $\quad (if \ n + sum\text{-}mset (snd \ \text{'\# } \{\#\ (a', -) \in \# \ b. \ a' = a\#\}) = 0 \ then \ \{\#\}$
 $\quad \quad \text{else } \{\#\ (a, n + sum\text{-}mset (snd \ \text{'\# } \{\#\ (a', -) \in \# \ b. \ a' = a\#\}))\#\} \rangle$
by (auto simp: *add-to-coefficient-def*)

interpretation *comp-fun-commute* $\langle add\text{-}to\text{-}coefficient \rangle$

proof –

```

have [iff]:
  ⟨a ≠ aa ⇒
    ((case x of (a', -) ⇒ a' = a) ∧ (case x of (a', -) ⇒ a' ≠ aa)) ↔
    (case x of (a', -) ⇒ a' = a)⟩ for a' aa a x
  by auto
show ⟨comp-fun-commute add-to-coefficient⟩
  unfolding add-to-coefficient-def
  by standard
  (auto intro!: ext simp: filter-filter-mset ac-simps add-eq-0-iff)
qed

```

```

lemma normalized-poly-normalize-poly[simp]:
  ⟨normalized-poly (normalize-poly p)⟩
  unfolding normalize-poly-def
  apply (induction p)
  subgoal by auto
  subgoal for x p
    by (cases x)
      (auto simp: add-to-coefficient-simps-If
        intro: normalized-poly-mono)
  done

```

7.2 Addition

inductive add-poly-p :: ⟨mset-polynomial × mset-polynomial × mset-polynomial ⇒ mset-polynomial × mset-polynomial × mset-polynomial ⇒ bool⟩ **where**

add-new-coeff-r:

⟨add-poly-p (p, add-mset x q, r) (p, q, add-mset x r)⟩ |

add-new-coeff-l:

⟨add-poly-p (add-mset x p, q, r) (p, q, add-mset x r)⟩ |

add-same-coeff-l:

⟨add-poly-p (add-mset (x, n) p, q, add-mset (x, m) r) (p, q, add-mset (x, n + m) r)⟩ |

add-same-coeff-r:

⟨add-poly-p (p, add-mset (x, n) q, add-mset (x, m) r) (p, q, add-mset (x, n + m) r)⟩ |

rem-0-coeff:

⟨add-poly-p (p, q, add-mset (x, 0) r) (p, q, r)⟩

inductive-cases add-poly-pE: ⟨add-poly-p S T⟩

lemmas add-poly-p-induct =

add-poly-p.induct[split-format(complete)]

lemma add-poly-p-empty-l:

⟨add-poly-p^{**} (p, q, r) ({#}, q, p + r)⟩

apply (induction p arbitrary: r)

subgoal **by** auto

subgoal

by (metis (no-types, lifting) add-new-coeff-l r-into-rtranclp

rtranclp-trans union-mset-add-mset-left union-mset-add-mset-right)

done

lemma add-poly-p-empty-r:

⟨add-poly-p^{**} (p, q, r) (p, {#}, q + r)⟩

apply (induction q arbitrary: r)

subgoal **by** auto

subgoal

by (metis (no-types, lifting) add-new-coeff-r r-into-rtranclp
 rtranclp-trans union-mset-add-mset-left union-mset-add-mset-right)
 done

lemma *add-poly-p-sym*:
 $\langle \text{add-poly-p } (p, q, r) (p', q', r') \longleftrightarrow \text{add-poly-p } (q, p, r) (q', p', r') \rangle$
 apply (rule iffI)
 subgoal
 by (cases rule: add-poly-p.cases, assumption)
 (auto intro: add-poly-p.intros)
 subgoal
 by (cases rule: add-poly-p.cases, assumption)
 (auto intro: add-poly-p.intros)
 done

lemma *wf-if-measure-in-wf*:
 $\langle \text{wf } R \implies (\bigwedge a b. (a, b) \in S \implies (\nu a, \nu b) \in R) \implies \text{wf } S \rangle$
 by (metis in-inv-image wfE-min wfI-min wf-inv-image)

lemma *lexn-n*:
 $\langle n > 0 \implies (x \# xs, y \# ys) \in \text{lexn } r \ n \longleftrightarrow$
 $(\text{length } xs = n-1 \wedge \text{length } ys = n-1) \wedge ((x, y) \in r \vee (x = y \wedge (xs, ys) \in \text{lexn } r (n-1))) \rangle$
 apply (cases n)
 apply simp
 by (auto simp: map-prod-def image-iff lex-prod-def)

lemma *wf-add-poly-p*:
 $\langle \text{wf } \{(x, y). \text{add-poly-p } y \ x\} \rangle$
 by (rule wf-if-measure-in-wf[where $R = \langle \text{lexn less-than } 3 \rangle$ and
 $\nu = \langle \lambda(a,b,c). [\text{size } a, \text{size } b, \text{size } c] \rangle$])
 (auto simp: add-poly-p.simps wf-lexn
 simp: lexn-n simp del: lexn.simps(2))

lemma *mult-poly-by-monom-simps[simp]*:
 $\langle \text{mult-poly-by-monom } t \ \{\#\} = \{\#\} \rangle$
 $\langle \text{mult-poly-by-monom } t \ (ps + qs) = \text{mult-poly-by-monom } t \ ps + \text{mult-poly-by-monom } t \ qs \rangle$
 $\langle \text{mult-poly-by-monom } a \ (\text{add-mset } p \ ps) = \text{add-mset } (\text{fst } a + \text{fst } p, \text{snd } a * \text{snd } p) \ (\text{mult-poly-by-monom } a \ ps) \rangle$

proof –

interpret *comp-fun-commute* $\langle (\lambda xs. \text{add-mset } (xs + t)) \rangle$ **for** t

by standard auto

show

$\langle \text{mult-poly-by-monom } t \ (ps + qs) = \text{mult-poly-by-monom } t \ ps + \text{mult-poly-by-monom } t \ qs \rangle$ **for** t

by (induction ps)

(auto simp: mult-poly-by-monom-def)

show

$\langle \text{mult-poly-by-monom } a \ (\text{add-mset } p \ ps) = \text{add-mset } (\text{fst } a + \text{fst } p, \text{snd } a * \text{snd } p) \ (\text{mult-poly-by-monom } a \ ps) \rangle$

$\langle \text{mult-poly-by-monom } t \ \{\#\} = \{\#\} \rangle$ **for** t

by (auto simp: mult-poly-by-monom-def)

qed

inductive *mult-poly-p* :: $\langle \text{mset-polynomial} \Rightarrow \text{mset-polynomial} \times \text{mset-polynomial} \Rightarrow \text{mset-polynomial} \times \text{mset-polynomial} \Rightarrow \text{bool} \rangle$

for q :: *mset-polynomial* **where**

mult-step:

$\langle \text{mult-poly-p } q \text{ (add-mset (xs, n) p, r) (p, (\lambda(ys, m). (\text{remdups-mset (xs + ys), n * m})) \text{ '# } q + r)) \rangle$

lemmas *mult-poly-p-induct* = *mult-poly-p.induct*[*split-format*(*complete*)]

7.3 Normalisation

inductive *normalize-poly-p* :: $\langle \text{mset-polynomial} \Rightarrow \text{mset-polynomial} \Rightarrow \text{bool} \rangle$ **where**

rem-0-coeff[*simp*, *intro*]:

$\langle \text{normalize-poly-p } p \text{ } q \Longrightarrow \text{normalize-poly-p (add-mset (xs, 0) p) } q \rangle$ |

merge-dup-coeff[*simp*, *intro*]:

$\langle \text{normalize-poly-p } p \text{ } q \Longrightarrow \text{normalize-poly-p (add-mset (xs, m) (add-mset (xs, n) p)) (add-mset (xs, m + n) q) \rangle$ |

same[*simp*, *intro*]:

$\langle \text{normalize-poly-p } p \text{ } p \rangle$ |

keep-coeff[*simp*, *intro*]:

$\langle \text{normalize-poly-p } p \text{ } q \Longrightarrow \text{normalize-poly-p (add-mset } x \text{ p) (add-mset } x \text{ q) \rangle$

7.4 Correctness

This locale maps string polynomials to real polynomials.

locale *poly-embed* =

fixes $\varphi :: \langle \text{string} \Rightarrow \text{nat} \rangle$

assumes $\varphi\text{-inj}: \langle \text{inj } \varphi \rangle$

begin

definition *poly-of-vars* :: $\text{term-poly} \Rightarrow ('a :: \{\text{comm-semiring-1}\}) \text{ mpoly}$ **where**

$\langle \text{poly-of-vars } xs = \text{fold-mset } (\lambda a \text{ b. Var } (\varphi a) * b) (1 :: 'a \text{ mpoly}) \text{ } xs \rangle$

lemma *poly-of-vars-simps*[*simp*]:

shows

$\langle \text{poly-of-vars (add-mset } x \text{ } xs) = \text{Var } (\varphi x) * (\text{poly-of-vars } xs :: ('a :: \{\text{comm-semiring-1}\}) \text{ mpoly}) \rangle$ **(is ?A)** **and**

$\langle \text{poly-of-vars (xs + ys)} = \text{poly-of-vars } xs * (\text{poly-of-vars } ys :: ('a :: \{\text{comm-semiring-1}\}) \text{ mpoly}) \rangle$ **(is ?B)**

proof –

interpret *comp-fun-commute* $\langle (\lambda a \text{ b. (b :: 'a :: \{\text{comm-semiring-1}\}) \text{ mpoly}) * \text{Var } (\varphi a) \rangle$

by *standard*

(auto simp: algebra-simps ac-simps

Var-def times-monomial-monomial intro!: ext)

show ?A

by *(auto simp: poly-of-vars-def comp-fun-commute-axioms fold-mset-fusion ac-simps)*

show ?B

apply *(auto simp: poly-of-vars-def ac-simps)*

by *(simp add: local.comp-fun-commute-axioms local.fold-mset-fusion semiring-normalization-rules(18))*

qed

definition *mononom-of-vars* **where**

$\langle \text{mononom-of-vars} \equiv (\lambda(xs, n). (+) (\text{Const } n * \text{poly-of-vars } xs)) \rangle$

interpretation *comp-fun-commute* \langle mononom-of-vars \rangle

by *standard*

(*auto simp: algebra-simps ac-simps mononom-of-vars-def*
Var-def times-monomial-monomial intro!: ext)

lemma [*simp*]:

\langle poly-of-vars $\{\#\} = 1$ \rangle

by (*auto simp: poly-of-vars-def*)

lemma *mononom-of-vars-add*[*simp*]:

\langle NO-MATCH $0\ b \implies$ mononom-of-vars $xs\ b = \text{Const}(\text{snd}\ xs) * \text{poly-of-vars}(\text{fst}\ xs) + b$ \rangle

by (*cases xs*)

(*auto simp: ac-simps mononom-of-vars-def*)

definition *polynomial-of-mset* :: \langle mset-polynomial \Rightarrow \rightarrow \rangle **where**

\langle polynomial-of-mset $p = \text{sum-mset}(\text{mononom-of-vars}\ \{\#\ p\}\ 0)$ \rangle

lemma *polynomial-of-mset-append*[*simp*]:

\langle polynomial-of-mset $(xs + ys) = \text{polynomial-of-mset}\ xs + \text{polynomial-of-mset}\ ys$ \rangle

by (*auto simp: ac-simps Const-def polynomial-of-mset-def*)

lemma *polynomial-of-mset-Cons*[*simp*]:

\langle polynomial-of-mset $(\text{add-mset}\ x\ ys) = \text{Const}(\text{snd}\ x) * \text{poly-of-vars}(\text{fst}\ x) + \text{polynomial-of-mset}\ ys$ \rangle

by (*cases x*)

(*auto simp: ac-simps polynomial-of-mset-def mononom-of-vars-def*)

lemma *polynomial-of-mset-empty*[*simp*]:

\langle polynomial-of-mset $\{\#\} = 0$ \rangle

by (*auto simp: polynomial-of-mset-def*)

lemma *polynomial-of-mset-mult-poly-by-monom*[*simp*]:

\langle polynomial-of-mset $(\text{mult-poly-by-monom}\ x\ ys) =$

$(\text{Const}(\text{snd}\ x) * \text{poly-of-vars}(\text{fst}\ x) * \text{polynomial-of-mset}\ ys)$ \rangle

by (*induction ys*)

(*auto simp: Const-mult algebra-simps*)

lemma *polynomial-of-mset-mult-poly-raw*[*simp*]:

\langle polynomial-of-mset $(\text{mult-poly-raw}\ xs\ ys) = \text{polynomial-of-mset}\ xs * \text{polynomial-of-mset}\ ys$ \rangle

unfolding *mult-poly-raw-def*

by (*induction xs arbitrary: ys*)

(*auto simp: Const-mult algebra-simps*)

lemma *polynomial-of-mset-uminus*:

\langle polynomial-of-mset $\{\#\ \text{case}\ x\ \text{of}\ (a, b) \Rightarrow (a, -b).\ x \in \#\ za\ \#\} =$

$- \text{polynomial-of-mset}\ za$ \rangle

by (*induction za*)

auto

lemma *X2-X-polynomial-bool-mult-in*:

\langle Var $(x1) * (\text{Var}\ (x1) * p) - \text{Var}\ (x1) * p \in \text{More-Modules.ideal}\ \text{polynomial-bool}$ \rangle

using *ideal-mult-right-in*[*OF X2-X-in-pac-ideal*[*of* $x1\ \langle\{\}\rangle$, *unfolded pac-ideal-def*], *of* p]

by (*auto simp: right-diff-distrib ac-simps power2-eq-square*)

lemma *polynomial-of-list-remove-powers-polynomial-bool*:
 $\langle \text{polynomial-of-mset } xs \rangle - \text{polynomial-of-mset } (\text{remove-powers } xs) \in \text{ideal polynomial-bool}$

proof (*induction xs*)
case *empty*
then show $\langle ?case \rangle$ **by** (*auto simp: remove-powers-def ideal.span-zero*)

next
case (*add x xs*)
have *H1*: $\langle x1 \in\# x2 \implies$
 $\text{Var } (\varphi x1) * \text{poly-of-vars } x2 - p \in \text{More-Modules.ideal polynomial-bool} \iff$
 $\text{poly-of-vars } x2 - p \in \text{More-Modules.ideal polynomial-bool}$
 \rangle **for** *x1 x2 p*
apply (*subst (2) ideal.span-add-eq[symmetric,*
of $\langle \text{Var } (\varphi x1) * \text{poly-of-vars } x2 - \text{poly-of-vars } x2 \rangle$ *)*)
apply (*drule multi-member-split*)
apply (*auto simp: X2-X-polynomial-bool-mult-in*)
done

have *diff*: $\langle \text{poly-of-vars } (x) - \text{poly-of-vars } (\text{remdups-mset } (x)) \in \text{ideal polynomial-bool} \rangle$ **for** *x*
by (*induction x*)
(auto simp: remove-powers-def ideal.span-zero H1
simp flip: right-diff-distrib intro!: ideal.span-scale)

have [*simp*]: $\langle \text{polynomial-of-mset } xs -$
 $\text{polynomial-of-mset } (\text{apfst remdups-mset } \# xs)$
 $\in \text{More-Modules.ideal polynomial-bool} \implies$
 $\text{poly-of-vars } ys * \text{poly-of-vars } ys -$
 $\text{poly-of-vars } ys * \text{poly-of-vars } (\text{remdups-mset } ys)$
 $\in \text{More-Modules.ideal polynomial-bool} \implies$
 $\text{polynomial-of-mset } xs + \text{Const } y * \text{poly-of-vars } ys -$
 $(\text{polynomial-of-mset } (\text{apfst remdups-mset } \# xs) +$
 $\text{Const } y * \text{poly-of-vars } (\text{remdups-mset } ys))$
 $\in \text{More-Modules.ideal polynomial-bool} \rangle$ **for** *y ys*
by (*metis add-diff-add diff ideal.scale-right-diff-distrib ideal.span-add ideal.span-scale*)

show *?case*
using *add*
apply (*cases x*)
subgoal for *ys y*
using *ideal-mult-right-in2[OF diff, of* $\langle \text{poly-of-vars } ys \rangle$ *ys]*
by (*auto simp: remove-powers-def right-diff-distrib*
ideal.span-diff ideal.span-add field-simps)
done

qed

lemma *add-poly-p-polynomial-of-mset*:
 $\langle \text{add-poly-p } (p, q, r) (p', q', r') \implies$
 $\text{polynomial-of-mset } r + (\text{polynomial-of-mset } p + \text{polynomial-of-mset } q) =$
 $\text{polynomial-of-mset } r' + (\text{polynomial-of-mset } p' + \text{polynomial-of-mset } q') \rangle$

apply (*induction rule: add-poly-p-induct*)
subgoal
by *auto*
subgoal
by *auto*
subgoal
by (*auto simp: algebra-simps Const-add*)
subgoal
by (*auto simp: algebra-simps Const-add*)

subgoal

by (*auto simp: algebra-simps Const-add*)

done

lemma *rtranclp-add-poly-p-polynomial-of-mset:*

$\langle \text{add-poly-p}^{**} (p, q, r) (p', q', r') \implies$

$\text{polynomial-of-mset } r + (\text{polynomial-of-mset } p + \text{polynomial-of-mset } q) =$
 $\text{polynomial-of-mset } r' + (\text{polynomial-of-mset } p' + \text{polynomial-of-mset } q') \rangle$

by (*induction rule: rtranclp-induct[of add-poly-p $\langle(-, -, -)\rangle \langle(-, -, -)\rangle$, split-format(complete), of for r]*)
(*auto dest: add-poly-p-polynomial-of-mset*)

lemma *rtranclp-add-poly-p-polynomial-of-mset-full:*

$\langle \text{add-poly-p}^{**} (p, q, \{\#\}) (\{\#\}, \{\#\}, r') \implies$

$\text{polynomial-of-mset } r' = (\text{polynomial-of-mset } p + \text{polynomial-of-mset } q) \rangle$

by (*drule rtranclp-add-poly-p-polynomial-of-mset*)

(*auto simp: ac-simps add-eq-0-iff*)

lemma *poly-of-vars-remdups-mset:*

$\langle \text{poly-of-vars } (\text{remdups-mset } (xs)) - (\text{poly-of-vars } xs)$

$\in \text{More-Modules.ideal polynomial-bool} \rangle$

apply (*induction xs*)

subgoal by (*auto simp: ideal.span-zero*)

subgoal for $x \ xs$

apply (*cases $\langle x \in \# \ xs \rangle$*)

apply (*metis (no-types, lifting) X2-X-polynomial-bool-mult-in diff-add-cancel diff-diff-eq2*
ideal.span-diff insert-DiffM poly-of-vars-simps(1) remdups-mset-singleton-sum)

by (*metis (no-types, lifting) ideal.span-scale poly-of-vars-simps(1) remdups-mset-singleton-sum*
right-diff-distrib)

done

lemma *polynomial-of-mset-mult-map:*

$\langle \text{polynomial-of-mset}$

$\{\#\text{case } x \text{ of } (ys, n) \Rightarrow (\text{remdups-mset } (ys + xs), n * m). x \in \# \ q\#\} -$
 $\text{Const } m * (\text{poly-of-vars } xs * \text{polynomial-of-mset } q)$

$\in \text{More-Modules.ideal polynomial-bool} \rangle$

(*is $\langle ?P \ q \in - \rangle$*)

proof (*induction q*)

case *empty*

then show *?case by* (*auto simp: algebra-simps ideal.span-zero*)

next

case (*add x q*)

then have $uP: \langle - ?P \ q \in \text{More-Modules.ideal polynomial-bool} \rangle$

using *ideal.span-neg by blast*

have $\langle \text{Const } b * (\text{Const } m * \text{poly-of-vars } (\text{remdups-mset } (a + xs))) -$
 $\text{Const } b * (\text{Const } m * (\text{poly-of-vars } a * \text{poly-of-vars } xs))$

$\in \text{More-Modules.ideal polynomial-bool} \rangle$ **for** $a \ b$

by (*auto simp: Const-mult simp flip: right-diff-distrib' poly-of-vars-simps*
intro!: ideal.span-scale poly-of-vars-remdups-mset)

then show *?case*

apply (*subst ideal.span-add-eq2[symmetric, OF uP]*)

apply (*cases x*)

apply (*auto simp: field-simps Const-mult simp flip:*
intro!: ideal.span-scale poly-of-vars-remdups-mset)

done

qed

lemma *mult-poly-p-mult-ideal*:

$\langle \text{mult-poly-p } q \text{ (} p, r \text{) (} p', r' \text{)} \implies$
 $(\text{polynomial-of-mset } p' * \text{polynomial-of-mset } q + \text{polynomial-of-mset } r') - (\text{polynomial-of-mset } p * \text{polynomial-of-mset } q + \text{polynomial-of-mset } r)$
 $\in \text{ideal polynomial-bool} \rangle$

proof (*induction rule: mult-poly-p-induct*)

case (*mult-step xs n p r*)

show *?case*

by (*auto simp: algebra-simps polynomial-of-mset-mult-map*)

qed

lemma *rtranclp-mult-poly-p-mult-ideal*:

$\langle (\text{mult-poly-p } q)^{**} (p, r) (p', r') \implies$
 $(\text{polynomial-of-mset } p' * \text{polynomial-of-mset } q + \text{polynomial-of-mset } r') - (\text{polynomial-of-mset } p * \text{polynomial-of-mset } q + \text{polynomial-of-mset } r)$
 $\in \text{ideal polynomial-bool} \rangle$

apply (*induction p' r' rule: rtranclp-induct[of $\langle \text{mult-poly-p } q \rangle \langle (p, r) \rangle \langle (p', q') \rangle$ for $p' q'$, split-format(complete)]*)

subgoal

by (*auto simp: ideal.span-zero*)

subgoal for *a b aa ba*

apply (*drule mult-poly-p-mult-ideal*)

apply (*drule ideal.span-add*)

apply *assumption*

by (*auto simp: group-add-class.diff-add-eq-diff-diff-swap*

add.inverse-distrib-swap ac-simps add-diff-eq

simp flip: diff-add-eq-diff-diff-swap)

done

lemma *rtranclp-mult-poly-p-mult-ideal-final*:

$\langle (\text{mult-poly-p } q)^{**} (p, \{\#\}) (\{\#\}, r) \implies$
 $(\text{polynomial-of-mset } r) - (\text{polynomial-of-mset } p * \text{polynomial-of-mset } q)$
 $\in \text{ideal polynomial-bool} \rangle$

by (*drule rtranclp-mult-poly-p-mult-ideal*) *auto*

lemma *normalize-poly-p-poly-of-mset*:

$\langle \text{normalize-poly-p } p \text{ } q \implies \text{polynomial-of-mset } p = \text{polynomial-of-mset } q \rangle$

apply (*induction rule: normalize-poly-p.induct*)

apply (*auto simp: Const-add algebra-simps*)

done

lemma *rtranclp-normalize-poly-p-poly-of-mset*:

$\langle \text{normalize-poly-p}^{**} p \text{ } q \implies \text{polynomial-of-mset } p = \text{polynomial-of-mset } q \rangle$

by (*induction rule: rtranclp-induct*)

(*auto simp: normalize-poly-p-poly-of-mset*)

end

It would be nice to have the property in the other direction too, but this requires a deep dive into the definitions of polynomials.

locale *poly-embed-bij* = *poly-embed* +

fixes *V N*

assumes φ -*bij*: $\langle \text{bij-betw } \varphi \text{ } V \text{ } N \rangle$

begin

definition $\varphi' :: \langle \text{nat} \Rightarrow \text{string} \rangle$ **where**

$\langle \varphi' = \text{the-inv-into } V \ \varphi \rangle$

lemma $\varphi'-\varphi[\text{simp}]$:

$\langle x \in V \Longrightarrow \varphi' (\varphi x) = x \rangle$

using φ -bij **unfolding** φ' -def

by (*meson bij-betw-imp-inj-on the-inv-into-f-f*)

lemma $\varphi-\varphi'[\text{simp}]$:

$\langle x \in N \Longrightarrow \varphi (\varphi' x) = x \rangle$

using φ -bij **unfolding** φ' -def

by (*meson f-the-inv-into-f-bij-betw*)

end

end

theory *PAC-Polynomials-Term*

imports *PAC-Polynomials*

Refine-Imperative-HOL.IICF

begin

8 Terms

We define some helper functions.

8.1 Ordering

lemma *fref-to-Down-curry-left*:

fixes $f :: \langle 'a \Rightarrow 'b \Rightarrow 'c \ \text{nres} \rangle$ **and**

$A :: \langle ('a \times 'b) \times 'd \rangle \ \text{set}$

shows

$\langle (\text{uncurry } f, g) \in [P]_f A \rightarrow \langle B \rangle \text{nres-rel} \Longrightarrow$

$(\bigwedge a \ b \ x'. P \ x' \Longrightarrow ((a, b), x') \in A \Longrightarrow f \ a \ b \leq \Downarrow B (g \ x')) \rangle$

unfolding *fref-def uncurry-def nres-rel-def*

by *auto*

lemma *fref-to-Down-curry-right*:

fixes $g :: \langle 'a \Rightarrow 'b \Rightarrow 'c \ \text{nres} \rangle$ **and** $f :: \langle 'd \Rightarrow - \ \text{nres} \rangle$ **and**

$A :: \langle 'd \times ('a \times 'b) \rangle \ \text{set}$

shows

$\langle (f, \text{uncurry } g) \in [P]_f A \rightarrow \langle B \rangle \text{nres-rel} \Longrightarrow$

$(\bigwedge a \ b \ x'. P (a, b) \Longrightarrow (x', (a, b)) \in A \Longrightarrow f \ x' \leq \Downarrow B (g \ a \ b)) \rangle$

unfolding *fref-def uncurry-def nres-rel-def*

by *auto*

type-synonym *term-poly-list* = $\langle \text{string list} \rangle$

type-synonym *llist-polynomial* = $\langle (\text{term-poly-list} \times \text{int}) \ \text{list} \rangle$

We instantiate the characters with typeclass `linorder` to be able to talk about sorted and so on.

definition *less-eq-char* :: $\langle \text{char} \Rightarrow \text{char} \Rightarrow \text{bool} \rangle$ **where**

$\langle \text{less-eq-char } c \ d = (((\text{of-char } c) :: \text{nat}) \leq \text{of-char } d) \rangle$

definition *less-char* :: $\langle \text{char} \Rightarrow \text{char} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{less-char } c \ d = (((\text{of-char } c) :: \text{nat}) < \text{of-char } d) \rangle$

global-interpretation *char*: *linorder less-eq-char less-char*
using *linorder-char*
unfolding *linorder-class-def class.linorder-def*
less-eq-char-def[symmetric] less-char-def[symmetric]
class.order-def order-class-def
class.preorder-def preorder-class-def
ord-class-def
apply *auto*
done

abbreviation *less-than-char* :: $\langle (\text{char} \times \text{char}) \text{ set} \rangle$ **where**
 $\langle \text{less-than-char} \equiv \text{p2rel less-char} \rangle$

lemma *less-than-char-def*:
 $\langle (x,y) \in \text{less-than-char} \longleftrightarrow \text{less-char } x \ y \rangle$
by (*auto simp: p2rel-def*)

lemma *trans-less-than-char[simp]*:
 $\langle \text{trans less-than-char} \rangle$ **and**
irrefl-less-than-char:
 $\langle \text{irrefl less-than-char} \rangle$ **and**
antisym-less-than-char:
 $\langle \text{antisym less-than-char} \rangle$
by (*auto simp: less-than-char-def trans-def irrefl-def antisym-def*)

8.2 Polynomials

definition *var-order-rel* :: $\langle (\text{string} \times \text{string}) \text{ set} \rangle$ **where**
 $\langle \text{var-order-rel} \equiv \text{lexord less-than-char} \rangle$

abbreviation *var-order* :: $\langle \text{string} \Rightarrow \text{string} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{var-order} \equiv \text{rel2p var-order-rel} \rangle$

abbreviation *term-order-rel* :: $\langle (\text{term-poly-list} \times \text{term-poly-list}) \text{ set} \rangle$ **where**
 $\langle \text{term-order-rel} \equiv \text{lexord var-order-rel} \rangle$

abbreviation *term-order* :: $\langle \text{term-poly-list} \Rightarrow \text{term-poly-list} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{term-order} \equiv \text{rel2p term-order-rel} \rangle$

definition *term-poly-list-rel* :: $\langle (\text{term-poly-list} \times \text{term-poly}) \text{ set} \rangle$ **where**
 $\langle \text{term-poly-list-rel} = \{(xs, ys). \$
 $\quad \text{ys} = \text{mset } xs \wedge$
 $\quad \text{distinct } xs \wedge$
 $\quad \text{sorted-wrt } (\text{rel2p var-order-rel}) \ xs\} \rangle$

definition *unsorted-term-poly-list-rel* :: $\langle (\text{term-poly-list} \times \text{term-poly}) \text{ set} \rangle$ **where**
 $\langle \text{unsorted-term-poly-list-rel} = \{(xs, ys). \$
 $\quad \text{ys} = \text{mset } xs \wedge \text{distinct } xs\} \rangle$

definition *poly-list-rel* :: $\langle \text{a} \Rightarrow ((\text{a} \times \text{int}) \text{ list} \times \text{mset-polynomial}) \text{ set} \rangle$ **where**
 $\langle \text{poly-list-rel } R = \{(xs, ys). \$

$(xs, ys) \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \wedge$
 $0 \notin \# \text{snd } \{\# \text{ys}\}$

definition *sorted-poly-list-rel-wrt* :: $\langle ('a \Rightarrow 'a \Rightarrow \text{bool})$
 $\Rightarrow ('a \times \text{string multiset}) \text{set} \Rightarrow (('a \times \text{int}) \text{list} \times \text{mset-polynomial}) \text{set} \rangle$ **where**
 $\langle \text{sorted-poly-list-rel-wrt } S \ R = \{(xs, ys).$
 $(xs, ys) \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \wedge$
 $\text{sorted-wrt } S \ (\text{map } \text{fst } xs) \wedge$
 $\text{distinct } (\text{map } \text{fst } xs) \wedge$
 $0 \notin \# \text{snd } \{\# \text{ys}\} \rangle$

abbreviation *sorted-poly-list-rel* **where**
 $\langle \text{sorted-poly-list-rel } R \equiv \text{sorted-poly-list-rel-wrt } R \ \text{term-poly-list-rel} \rangle$

abbreviation *sorted-poly-rel* **where**
 $\langle \text{sorted-poly-rel} \equiv \text{sorted-poly-list-rel } \text{term-order} \rangle$

definition *sorted-repeat-poly-list-rel-wrt* :: $\langle ('a \Rightarrow 'a \Rightarrow \text{bool})$
 $\Rightarrow ('a \times \text{string multiset}) \text{set} \Rightarrow (('a \times \text{int}) \text{list} \times \text{mset-polynomial}) \text{set} \rangle$ **where**
 $\langle \text{sorted-repeat-poly-list-rel-wrt } S \ R = \{(xs, ys).$
 $(xs, ys) \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \wedge$
 $\text{sorted-wrt } S \ (\text{map } \text{fst } xs) \wedge$
 $0 \notin \# \text{snd } \{\# \text{ys}\} \rangle$

abbreviation *sorted-repeat-poly-list-rel* **where**
 $\langle \text{sorted-repeat-poly-list-rel } R \equiv \text{sorted-repeat-poly-list-rel-wrt } R \ \text{term-poly-list-rel} \rangle$

abbreviation *sorted-repeat-poly-rel* **where**
 $\langle \text{sorted-repeat-poly-rel} \equiv \text{sorted-repeat-poly-list-rel } (\text{rel2p } (\text{Id} \cup \text{lexord } \text{var-order-rel})) \rangle$

abbreviation *unsorted-poly-rel* **where**
 $\langle \text{unsorted-poly-rel} \equiv \text{poly-list-rel } \text{term-poly-list-rel} \rangle$

lemma *sorted-poly-list-rel-empty-l[simp]*:
 $\langle (\[], s') \in \text{sorted-poly-list-rel-wrt } S \ T \longleftrightarrow s' = \{\#\} \rangle$
by (cases s')
(auto simp: sorted-poly-list-rel-wrt-def list-mset-rel-def br-def)

definition *fully-unsorted-poly-list-rel* :: $\langle \text{set} \Rightarrow (('a \times \text{int}) \text{list} \times \text{mset-polynomial}) \text{set} \rangle$ **where**
 $\langle \text{fully-unsorted-poly-list-rel } R = \{(xs, ys).$
 $(xs, ys) \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \rangle$

abbreviation *fully-unsorted-poly-rel* **where**
 $\langle \text{fully-unsorted-poly-rel} \equiv \text{fully-unsorted-poly-list-rel } \text{unsorted-term-poly-list-rel} \rangle$

lemma *fully-unsorted-poly-list-rel-empty-iff[simp]*:
 $\langle (p, \{\#\}) \in \text{fully-unsorted-poly-list-rel } R \longleftrightarrow p = [] \rangle$
 $\langle (\[], p') \in \text{fully-unsorted-poly-list-rel } R \longleftrightarrow p' = \{\#\} \rangle$
by (auto simp: fully-unsorted-poly-list-rel-def list-mset-rel-def br-def)

definition *poly-list-rel-with0* :: $\langle \text{set} \Rightarrow (('a \times \text{int}) \text{list} \times \text{mset-polynomial}) \text{set} \rangle$ **where**

$\langle \text{poly-list-rel-with0 } R = \{(xs, ys). \\ (xs, ys) \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{list-mset-rel} \rangle$

abbreviation *unsorted-poly-rel-with0* **where**

$\langle \text{unsorted-poly-rel-with0} \equiv \text{fully-unsorted-poly-list-rel term-poly-list-rel} \rangle$

lemma *poly-list-rel-with0-empty-iff[simp]*:

$\langle (p, \{\#\}) \in \text{poly-list-rel-with0 } R \longleftrightarrow p = [] \rangle$

$\langle ([], p') \in \text{poly-list-rel-with0 } R \longleftrightarrow p' = \{\#\} \rangle$

by (*auto simp: poly-list-rel-with0-def list-mset-rel-def br-def*)

definition *sorted-repeat-poly-list-rel-with0-wrt* :: $\langle 'a \Rightarrow 'a \Rightarrow \text{bool} \rangle$

$\Rightarrow ('a \times \text{string multiset}) \text{set} \Rightarrow (('a \times \text{int}) \text{list} \times \text{mset-polynomial}) \text{set} \rangle$ **where**

$\langle \text{sorted-repeat-poly-list-rel-with0-wrt } S R = \{(xs, ys). \\ (xs, ys) \in \langle R \times_r \text{int-rel} \rangle \text{list-rel } O \text{list-mset-rel} \wedge \\ \text{sorted-wrt } S (\text{map fst } xs) \rangle$

abbreviation *sorted-repeat-poly-list-rel-with0* **where**

$\langle \text{sorted-repeat-poly-list-rel-with0 } R \equiv \text{sorted-repeat-poly-list-rel-with0-wrt } R \text{ term-poly-list-rel} \rangle$

abbreviation *sorted-repeat-poly-rel-with0* **where**

$\langle \text{sorted-repeat-poly-rel-with0} \equiv \text{sorted-repeat-poly-list-rel-with0 } (\text{rel2p } (\text{Id} \cup \text{lexord var-order-rel})) \rangle$

lemma *term-poly-list-relD*:

$\langle (xs, ys) \in \text{term-poly-list-rel} \implies \text{distinct } xs \rangle$

$\langle (xs, ys) \in \text{term-poly-list-rel} \implies ys = \text{mset } xs \rangle$

$\langle (xs, ys) \in \text{term-poly-list-rel} \implies \text{sorted-wrt } (\text{rel2p var-order-rel}) xs \rangle$

$\langle (xs, ys) \in \text{term-poly-list-rel} \implies \text{sorted-wrt } (\text{rel2p } (\text{Id} \cup \text{var-order-rel})) xs \rangle$

apply (*auto simp: term-poly-list-rel-def; fail*)⁺

by (*metis (mono-tags, lifting) CollectD UnI2 rel2p-def sorted-wrt-mono-rel split-conv term-poly-list-rel-def*)

end

theory *PAC-Polynomials-Operations*

imports *PAC-Polynomials-Term PAC-Checker-Specification*

begin

8.3 Addition

In this section, we refine the polynomials to list. These lists will be used in our checker to represent the polynomials and execute operations.

There is one *key* difference between the list representation and the usual representation: in the former, coefficients can be zero and monomials can appear several times. This makes it easier to reason on intermediate representation where this has not yet been sanitized.

fun *add-poly-l'* :: $\langle \text{llist-polynomial} \times \text{llist-polynomial} \Rightarrow \text{llist-polynomial} \rangle$ **where**

$\langle \text{add-poly-l}' (p, []) = p \rangle \mid$

$\langle \text{add-poly-l}' ([], q) = q \rangle \mid$

$\langle \text{add-poly-l}' ((xs, n) \# p, (ys, m) \# q) =$

$(\text{if } xs = ys \text{ then if } n + m = 0 \text{ then } \text{add-poly-l}' (p, q) \text{ else}$

$\text{let } pq = \text{add-poly-l}' (p, q) \text{ in}$

$((xs, n + m) \# pq)$

$\text{else if } (xs, ys) \in \text{term-order-rel}$

then

```

      let pq = add-poly-l' (p, (ys, m) # q) in
      ((xs, n) # pq)
    else
      let pq = add-poly-l' ((xs, n) # p, q) in
      ((ys, m) # pq)
  )>

```

definition *add-poly-l* :: $\langle \text{llist-polynomial} \times \text{llist-polynomial} \Rightarrow \text{llist-polynomial nres} \rangle$ **where**

```

<add-poly-l = RECT
  (λadd-poly-l (p, q).
    case (p,q) of
      (p, []) ⇒ RETURN p
    | ([], q) ⇒ RETURN q
    | ((xs, n) # p, (ys, m) # q) ⇒
      (if xs = ys then if n + m = 0 then add-poly-l (p, q) else
        do {
          pq ← add-poly-l (p, q);
          RETURN ((xs, n + m) # pq)
        }
      else if (xs, ys) ∈ term-order-rel
        then do {
          pq ← add-poly-l (p, (ys, m) # q);
          RETURN ((xs, n) # pq)
        }
      else do {
          pq ← add-poly-l ((xs, n) # p, q);
          RETURN ((ys, m) # pq)
        }
      )))>

```

definition *nonzero-coeffs* **where**

```

<nonzero-coeffs a ⇔ 0 ∉ # snd '# a>

```

lemma *nonzero-coeffs-simps[simp]*:

```

<nonzero-coeffs {#}>
<nonzero-coeffs (add-mset (xs, n) a) ⇔ nonzero-coeffs a ∧ n ≠ 0>
by (auto simp: nonzero-coeffs-def)

```

lemma *nonzero-coeffsD*:

```

<nonzero-coeffs a ⇒ (x, n) ∈ # a ⇒ n ≠ 0>
by (auto simp: nonzero-coeffs-def)

```

lemma *sorted-poly-list-rel-ConsD*:

```

<((ys, n) # p, a) ∈ sorted-poly-list-rel S ⇒ (p, remove1-mset (mset ys, n) a) ∈ sorted-poly-list-rel S
^

```

```

  (mset ys, n) ∈ # a ∧ (∀ x ∈ set p. S ys (fst x)) ∧ sorted-wrt (rel2p var-order-rel) ys ∧
  distinct ys ∧ ys ∉ set (map fst p) ∧ n ≠ 0 ∧ nonzero-coeffs a>

```

unfolding *sorted-poly-list-rel-wrt-def prod.case mem-Collect-eq list-rel-def*

apply (*clarsimp*)

apply (*subst (asm) list.rel-sel*)

apply (*intro conjI*)

apply (*rename-tac y, rule-tac b = <tl y> in relcompI*)

apply (*auto simp: sorted-poly-list-rel-wrt-def list-mset-rel-def br-def list.tl-def term-poly-list-rel-def nonzero-coeffs-def split: list.splits*)

done

lemma *sorted-poly-list-rel-Cons-iff*:

$\langle (ys, n) \# p, a \rangle \in \text{sorted-poly-list-rel } S \iff (p, \text{remove1-mset } (\text{mset } ys, n) a) \in \text{sorted-poly-list-rel } S$
 \wedge

$(\text{mset } ys, n) \in \# a \wedge (\forall x \in \text{set } p. S \text{ } ys \text{ } (\text{fst } x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } ys \wedge$
 $\text{distinct } ys \wedge ys \notin \text{set } (\text{map } \text{fst } p) \wedge n \neq 0 \wedge \text{nonzero-coeffs } a$

apply (*rule iffI*)

subgoal

by (*auto dest!: sorted-poly-list-rel-ConsD*)

subgoal

unfolding *sorted-poly-list-rel-wrt-def prod.case mem-Collect-eq*
list-rel-def

apply (*clarsimp*)

apply (*intro conjI*)

apply (*rename-tac y; rule-tac b = $\langle \text{mset } ys, n \rangle \# y$ in relcompI*)

by (*auto simp: sorted-poly-list-rel-wrt-def list-mset-rel-def br-def*
term-poly-list-rel-def add-mset-eq-add-mset eq-commute[of - $\langle \text{mset } \rightarrow$]
nonzero-coeffs-def
dest!: multi-member-split)

done

lemma *sorted-repeat-poly-list-rel-ConsD*:

$\langle (ys, n) \# p, a \rangle \in \text{sorted-repeat-poly-list-rel } S \implies (p, \text{remove1-mset } (\text{mset } ys, n) a) \in \text{sorted-repeat-poly-list-rel } S$
 \wedge

$(\text{mset } ys, n) \in \# a \wedge (\forall x \in \text{set } p. S \text{ } ys \text{ } (\text{fst } x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } ys \wedge$
 $\text{distinct } ys \wedge n \neq 0 \wedge \text{nonzero-coeffs } a$

unfolding *sorted-repeat-poly-list-rel-wrt-def prod.case mem-Collect-eq*
list-rel-def

apply (*clarsimp*)

apply (*subst (asm) list.rel-sel*)

apply (*intro conjI*)

apply (*rename-tac y, rule-tac b = $\langle \text{tl } y \rangle$ in relcompI*)

apply (*auto simp: sorted-poly-list-rel-wrt-def list-mset-rel-def br-def*
list.tl-def term-poly-list-rel-def nonzero-coeffs-def split: list.splits)

done

lemma *sorted-repeat-poly-list-rel-Cons-iff*:

$\langle (ys, n) \# p, a \rangle \in \text{sorted-repeat-poly-list-rel } S \iff (p, \text{remove1-mset } (\text{mset } ys, n) a) \in \text{sorted-repeat-poly-list-rel } S$
 \wedge

$(\text{mset } ys, n) \in \# a \wedge (\forall x \in \text{set } p. S \text{ } ys \text{ } (\text{fst } x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } ys \wedge$
 $\text{distinct } ys \wedge n \neq 0 \wedge \text{nonzero-coeffs } a$

apply (*rule iffI*)

subgoal

by (*auto dest!: sorted-repeat-poly-list-rel-ConsD*)

subgoal

unfolding *sorted-repeat-poly-list-rel-wrt-def prod.case mem-Collect-eq*
list-rel-def

apply (*clarsimp*)

apply (*intro conjI*)

apply (*rename-tac y, rule-tac b = $\langle \text{mset } ys, n \rangle \# y$ in relcompI*)

by (*auto simp: sorted-repeat-poly-list-rel-wrt-def list-mset-rel-def br-def*
term-poly-list-rel-def add-mset-eq-add-mset eq-commute[of - $\langle \text{mset } \rightarrow$]
nonzero-coeffs-def)

dest!: multi-member-split)
done

lemma *add-poly-p-add-mset-sum-0*:

$\langle n + m = 0 \implies \text{add-poly-p}^{**} (A, Aa, \{\#\}) (\{\#\}, \{\#\}, r) \implies$
 add-poly-p^{**}
 $(\text{add-mset} (\text{mset } ys, n) A, \text{add-mset} (\text{mset } ys, m) Aa, \{\#\})$
 $(\{\#\}, \{\#\}, r) \rangle$

apply (*rule converse-rtranclp-into-rtranclp*)

apply (*rule add-poly-p.add-new-coeff-r*)

apply (*rule converse-rtranclp-into-rtranclp*)

apply (*rule add-poly-p.add-same-coeff-l*)

apply (*rule converse-rtranclp-into-rtranclp*)

apply (*auto intro: add-poly-p.rem-0-coeff*)

done

lemma *monoms-add-poly-l'D*:

$\langle (aa, ba) \in \text{set} (\text{add-poly-l}' x) \implies aa \in \text{fst ' set} (\text{fst } x) \vee aa \in \text{fst ' set} (\text{snd } x) \rangle$

by (*induction x rule: add-poly-l'.induct*)

(*auto split: if-splits*)

lemma *add-poly-p-add-to-result*:

$\langle \text{add-poly-p}^{**} (A, B, r) (A', B', r') \implies$
 add-poly-p^{**}
 $(A, B, p + r) (A', B', p + r') \rangle$

apply (*induction rule: rtranclp-induct[of add-poly-p $\langle(-, -, -)\rangle \langle(-, -, -)\rangle$, split-format(complete), of for r]*)

subgoal by auto

by (*elim add-poly-pE*)

(*metis (no-types, lifting) Pair-inject add-poly-p.intros rtranclp.simps union-mset-add-mset-right*) $+$

lemma *add-poly-p-add-mset-comb*:

$\langle \text{add-poly-p}^{**} (A, Aa, \{\#\}) (\{\#\}, \{\#\}, r) \implies$
 add-poly-p^{**}
 $(\text{add-mset} (xs, n) A, Aa, \{\#\})$
 $(\{\#\}, \{\#\}, \text{add-mset} (xs, n) r) \rangle$

apply (*rule converse-rtranclp-into-rtranclp*)

apply (*rule add-poly-p.add-new-coeff-l*)

using *add-poly-p-add-to-result*[of $A Aa \langle\{\#\}\rangle \langle\{\#\}\rangle \langle\{\#\}\rangle r \langle\{\#\}(xs, n)\#\rangle$]

by auto

lemma *add-poly-p-add-mset-comb2*:

$\langle \text{add-poly-p}^{**} (A, Aa, \{\#\}) (\{\#\}, \{\#\}, r) \implies$
 add-poly-p^{**}
 $(\text{add-mset} (ys, n) A, \text{add-mset} (ys, m) Aa, \{\#\})$
 $(\{\#\}, \{\#\}, \text{add-mset} (ys, n + m) r) \rangle$

apply (*rule converse-rtranclp-into-rtranclp*)

apply (*rule add-poly-p.add-new-coeff-r*)

apply (*rule converse-rtranclp-into-rtranclp*)

apply (*rule add-poly-p.add-same-coeff-l*)

using *add-poly-p-add-to-result*[of $A Aa \langle\{\#\}\rangle \langle\{\#\}\rangle \langle\{\#\}\rangle r \langle\{\#\}(ys, n+m)\#\rangle$]

by auto

lemma *add-poly-p-add-mset-comb3*:
 $\langle \text{add-poly-p}^{**} (A, Aa, \{\#\}) (\{\#\}, \{\#\}, r) \implies$
 add-poly-p^{**}
 $(A, \text{add-mset } (ys, m) Aa, \{\#\})$
 $(\{\#\}, \{\#\}, \text{add-mset } (ys, m) r) \rangle$
apply (*rule converse-rtranclp-into-rtranclp*)
apply (*rule add-poly-p.add-new-coeff-r*)
using *add-poly-p-add-to-result*[of $A Aa \langle \{\#\} \rangle \langle \{\#\} \rangle \langle \{\#\} \rangle r \langle \{\#(ys, m)\# \}$]
by *auto*

lemma *total-on-lexord*:
 $\langle \text{Relation.total-on UNIV } R \implies \text{Relation.total-on UNIV } (\text{lexord } R) \rangle$
apply (*auto simp: Relation.total-on-def*)
by (*meson lexord-linear*)

lemma *antisym-lexord*:
 $\langle \text{antisym } R \implies \text{irrefl } R \implies \text{antisym } (\text{lexord } R) \rangle$
by (*auto simp: antisym-def lexord-def irrefl-def*
elim!: list-match-lcl-lcl)

lemma *less-than-char-linear*:
 $\langle (a, b) \in \text{less-than-char} \vee$
 $a = b \vee (b, a) \in \text{less-than-char} \rangle$
by (*auto simp: less-than-char-def*)

lemma *total-on-lexord-less-than-char-linear*:
 $\langle xs \neq ys \implies (xs, ys) \notin \text{lexord } (\text{lexord less-than-char}) \longleftrightarrow$
 $(ys, xs) \in \text{lexord } (\text{lexord less-than-char}) \rangle$
using *lexord-linear*[of $\langle \text{lexord less-than-char} \rangle xs ys]$
using *lexord-linear*[of $\langle \text{less-than-char} \rangle$] *less-than-char-linear*
using *lexord-irrefl*[OF *irrefl-less-than-char*]
antisym-lexord[OF *antisym-lexord*[OF *antisym-less-than-char irrefl-less-than-char*]]
apply (*auto simp: antisym-def Relation.total-on-def*)
done

lemma *sorted-poly-list-rel-nonzeroD*:
 $\langle (p, r) \in \text{sorted-poly-list-rel term-order} \implies$
 $\text{nonzero-coeffs } (r) \rangle$
 $\langle (p, r) \in \text{sorted-poly-list-rel } (\text{rel2p } (\text{lexord } (\text{lexord less-than-char}))) \implies$
 $\text{nonzero-coeffs } (r) \rangle$
by (*auto simp: sorted-poly-list-rel-wrt-def nonzero-coeffs-def*)

lemma *add-poly-l'-add-poly-p*:
assumes $\langle (pq, pq') \in \text{sorted-poly-rel} \times_r \text{sorted-poly-rel} \rangle$
shows $\langle \exists r. (\text{add-poly-l}' pq, r) \in \text{sorted-poly-rel} \wedge$
 $\text{add-poly-p}^{**} (\text{fst } pq', \text{snd } pq', \{\#\}) (\{\#\}, \{\#\}, r) \rangle$
supply [[*goals-limit=1*]]
using *assms*
apply (*induction* $\langle pq \rangle$ *arbitrary: pq'* *rule: add-poly-l'.induct*)
subgoal for $p pq'$
using *add-poly-p-empty-l*[of $\langle \text{fst } pq' \rangle \langle \{\#\} \rangle \langle \{\#\} \rangle$]
by (*cases* pq') (*auto intro!: exI*[of $-\langle \text{fst } pq' \rangle$])
subgoal for $x p pq'$
using *add-poly-p-empty-r*[of $\langle \{\#\} \rangle \langle \text{snd } pq' \rangle \langle \{\#\} \rangle$]


```

  by (cases pq') (auto intro!: exI[of - ⟨snd pq'⟩])
subgoal premises p for xs n p ys m q pq'
  apply (cases pq') — Isabelle does a completely stupid case distinction here
  apply (cases ⟨xs = ys⟩)
  subgoal
    apply (cases ⟨n + m = 0⟩)
    subgoal
      using p(1)[of ⟨(remove1-mset (mset xs, n) (fst pq'), remove1-mset (mset ys, m) (snd pq'))⟩]
p(5-)
      apply (auto dest!: sorted-poly-list-rel-ConsD multi-member-split
            )
      using add-poly-p-add-mset-sum-0 by blast
    subgoal
      using p(2)[of ⟨(remove1-mset (mset xs, n) (fst pq'), remove1-mset (mset ys, m) (snd pq'))⟩]
p(5-)
      apply (auto dest!: sorted-poly-list-rel-ConsD multi-member-split)
      apply (rule-tac x = ⟨add-mset (mset ys, n + m) r⟩ in exI)
      apply (fastforce dest!: monoms-add-poly-l'D simp: sorted-poly-list-rel-Cons-iff rel2p-def
            sorted-poly-list-rel-nonzeroD var-order-rel-def
            intro: add-poly-p-add-mset-comb2)
      done
    done
  done
subgoal
  apply (cases ⟨(xs, ys) ∈ term-order-rel⟩)
  subgoal
    using p(3)[of ⟨(remove1-mset (mset xs, n) (fst pq'), (snd pq'))⟩] p(5-)
    apply (auto dest!: multi-member-split simp: sorted-poly-list-rel-Cons-iff rel2p-def)
    apply (rule-tac x = ⟨add-mset (mset xs, n) r⟩ in exI)
    apply (auto dest!: monoms-add-poly-l'D)
    apply (auto intro: lexord-trans add-poly-p-add-mset-comb simp: lexord-transI var-order-rel-def)
    apply (rule lexord-trans)
    apply assumption
    apply (auto intro: lexord-trans add-poly-p-add-mset-comb simp: lexord-transI
          sorted-poly-list-rel-nonzeroD var-order-rel-def)
    using total-on-lexord-less-than-char-linear by fastforce

  subgoal
    using p(4)[of ⟨(fst pq', remove1-mset (mset ys, m) (snd pq'))⟩] p(5-)
    apply (auto dest!: multi-member-split simp: sorted-poly-list-rel-Cons-iff rel2p-def
          var-order-rel-def)
    apply (rule-tac x = ⟨add-mset (mset ys, m) r⟩ in exI)
    apply (auto dest!: monoms-add-poly-l'D
          simp: total-on-lexord-less-than-char-linear)
    apply (auto intro: lexord-trans add-poly-p-add-mset-comb simp: lexord-transI
          total-on-lexord-less-than-char-linear var-order-rel-def)
    apply (rule lexord-trans)
    apply assumption
    apply (auto intro: lexord-trans add-poly-p-add-mset-comb3 simp: lexord-transI
          sorted-poly-list-rel-nonzeroD var-order-rel-def)
    using total-on-lexord-less-than-char-linear by fastforce
  done
done
done
done

```

lemma *add-poly-l-add-poly*:

⟨*add-poly-l* $x = \text{RETURN } (\text{add-poly-l}' x)$ ⟩

unfolding *add-poly-l-def*

by (*induction* x *rule*: *add-poly-l'.induct*)

(*solves* ⟨*subst* *RECT-unfold*, *refine-mono*, *simp split: list.split*⟩)+

lemma *add-poly-l-spec*:

⟨(*add-poly-l*, *uncurry* ($\lambda p q. \text{SPEC}(\lambda r. \text{add-poly-p}^{**} (p, q, \{\#\}) (\{\#\}, \{\#\}, r))$)) \in
sorted-poly-rel \times_r *sorted-poly-rel* \rightarrow_f ⟨*sorted-poly-rel*⟩*nres-rel*⟩

unfolding *add-poly-l-add-poly*

apply (*intro nres-relI* *frefI*)

apply (*drule* *add-poly-l'-add-poly-p*)

apply (*auto simp: conc-fun-RES*)

done

definition *sort-poly-spec* :: ⟨*l**list-polynomial* \Rightarrow *l**list-polynomial nres*⟩ **where**

⟨*sort-poly-spec* $p =$

SPEC($\lambda p'. \text{mset } p = \text{mset } p' \wedge \text{sorted-wrt } (\text{rel2p } (\text{Id} \cup \text{term-order-rel})) (\text{map fst } p')$ ⟩)

lemma *sort-poly-spec-id*:

assumes ⟨ $(p, p') \in \text{unsorted-poly-rel}$ ⟩

shows ⟨*sort-poly-spec* $p \leq \Downarrow$ (*sorted-repeat-poly-rel*) (*RETURN* p')⟩

proof –

obtain y **where**

py : ⟨ $(p, y) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel}$ ⟩ **and**

$p'-y$: ⟨ $p' = \text{mset } y$ ⟩ **and**

$zero$: ⟨ $0 \notin \# \text{snd } \# p'$ ⟩

using *assms*

unfolding *sort-poly-spec-def* *poly-list-rel-def* *sorted-poly-list-rel-wrt-def*

by (*auto simp: list-mset-rel-def* *br-def*)

then have [*simp*]: ⟨*length* $y = \text{length } p$ ⟩

by (*auto simp: list-rel-def* *list-all2-conv-all-nth*)

have H : ⟨ (x, p')

$\in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel}$ O *list-mset-rel*⟩

if px : ⟨ $\text{mset } p = \text{mset } x$ ⟩ **and** ⟨*sorted-wrt* (*rel2p* (*Id* \cup *lexord var-order-rel*)) (*map fst* x)⟩

for x :: ⟨*l**list-polynomial*⟩

proof –

from px **have** ⟨*length* $x = \text{length } p$ ⟩

by (*metis* *size-mset*)

from px **have** ⟨*mset* $x = \text{mset } p$ ⟩

by *simp*

then obtain f **where** ⟨ f *permutes* $\{..<\text{length } p\}$ ⟩ ⟨*permute-list* $f p = x$ ⟩

by (*rule* *mset-eq-permutation*)

with ⟨*length* $x = \text{length } p$ ⟩ **have** f : ⟨*bij-betw* $f \{..<\text{length } x\} \{..<\text{length } p\}$ ⟩

by (*simp* *add: permutes-imp-bij*)

from ⟨ f *permutes* $\{..<\text{length } p\}$ ⟩ ⟨*permute-list* $f p = x$ ⟩ [*symmetric*]

have [*simp*]: ⟨ $\bigwedge i. i < \text{length } x \implies x ! i = p ! (f i)$ ⟩

by (*simp* *add: permute-list-nth*)

let $?y = \langle \text{map } (\lambda i. y ! f i) [0 ..< \text{length } x] \rangle$

have ⟨ $i < \text{length } y \implies (p ! f i, y ! f i) \in \text{term-poly-list-rel} \times_r \text{int-rel}$ ⟩ **for** i

using *list-all2-nthD*[*of* - $p y$

⟨ $f i$ ⟩, *OF* py [*unfolded* *list-rel-def* *mem-Collect-eq* *prod.case*]]

mset-eq-length[*OF* px] f

by (*auto simp: list-rel-def* *list-all2-conv-all-nth* *bij-betw-def*)

then have ⟨ $(x, ?y) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel}$ ⟩ **and**

```

xy: ⟨length x = length y⟩
using py list-all2-nthD[of ⟨rel2p (term-poly-list-rel ×r int-rel)⟩ p y
  ⟨f i⟩ for i, simplified] mset-eq-length[OF px]
by (auto simp: list-rel-def list-all2-conv-all-nth)
moreover {
  have f: ⟨mset-set {0..using f mset-eq-length[OF px]
    by (auto simp: bij-betw-def lessThan-atLeast0 image-mset-mset-set)
  have ⟨mset y = {#y ! f x. x ∈# mset-set {0..by (subst drop-0[symmetric], subst mset-drop-upto, subst xy[symmetric], subst f)
    auto
  then have ⟨(?y, p') ∈ list-mset-rel⟩
    by (auto simp: list-mset-rel-def br-def p'-y)
}
ultimately show ?thesis
by (auto intro!: relcompI[of - ?y])
qed
show ?thesis
using zero
unfolding sort-poly-spec-def poly-list-rel-def sorted-repeat-poly-list-rel-wrt-def
by refine-rcg (auto intro: H)
qed

```

8.4 Multiplication

```

fun mult-monoms :: ⟨term-poly-list ⇒ term-poly-list ⇒ term-poly-list⟩ where
  ⟨mult-monoms p [] = p⟩ |
  ⟨mult-monoms [] p = p⟩ |
  ⟨mult-monoms (x # p) (y # q) =
    (if x = y then x # mult-monoms p q
     else if (x, y) ∈ var-order-rel then x # mult-monoms p (y # q)
     else y # mult-monoms (x # p) q)⟩

```

```

lemma term-poly-list-rel-empty-iff[simp]:
  ⟨([], q') ∈ term-poly-list-rel ⟷ q' = {#}⟩
by (auto simp: term-poly-list-rel-def)

```

```

lemma mset-eqD-set-mset: ⟨mset xs = A ⟹ set xs = set-mset A⟩
by auto

```

```

lemma term-poly-list-rel-Cons-iff:
  ⟨(y # p, p') ∈ term-poly-list-rel ⟷
    (p, remove1-mset y p') ∈ term-poly-list-rel ∧
    y ∈# p' ∧ y ∉ set p ∧ y ∉# remove1-mset y p' ∧
    (∀ x ∈# mset p. (y, x) ∈ var-order-rel)⟩
by (auto simp: term-poly-list-rel-def rel2p-def dest!: multi-member-split mset-eqD-set-mset)

```

```

lemma var-order-rel-antisym[simp]:
  ⟨(y, y) ∉ var-order-rel⟩
by (simp add: less-than-char-def lexord-irreflexive var-order-rel-def)

```

```

lemma term-poly-list-rel-remdups-mset:
  ⟨(p, p') ∈ term-poly-list-rel ⟹
    (p, remdups-mset p') ∈ term-poly-list-rel⟩
by (auto simp: term-poly-list-rel-def distinct-mset-remdups-mset-id simp flip: distinct-mset-mset-distinct)

```

lemma *var-notin-notin-mult-monomsD*:

$\langle y \in \text{set } (\text{mult-monoms } p \ q) \implies y \in \text{set } p \vee y \in \text{set } q \rangle$

by (*induction* $p \ q$ *arbitrary*: $p' \ q'$ *rule*: *mult-monoms.induct*) (*auto simp*: *if-splits*)

lemma *term-poly-list-rel-set-mset*:

$\langle (p, q) \in \text{term-poly-list-rel} \implies \text{set } p = \text{set-mset } q \rangle$

by (*auto simp*: *term-poly-list-rel-def*)

lemma *mult-monoms-spec*:

$\langle (\text{mult-monoms}, (\lambda a \ b. \text{remdups-mset } (a + b))) \in \text{term-poly-list-rel} \rightarrow \text{term-poly-list-rel} \rightarrow \text{term-poly-list-rel} \rangle$

proof –

have [*dest*]: $\langle \text{remdups-mset } (A + Aa) = \text{add-mset } y \ Ab \implies y \notin \# A \implies$

$y \notin \# Aa \implies$

$\text{False} \rangle$ **for** $Aa \ Ab \ y \ A$

by (*metis* *set-mset-remdups-mset union-iff union-single-eq-member*)

show *?thesis*

apply (*intro fun-relI*)

apply (*rename-tac* $p \ p' \ q \ q'$)

subgoal **for** $p \ p' \ q \ q'$

apply (*induction* $p \ q$ *arbitrary*: $p' \ q'$ *rule*: *mult-monoms.induct*)

subgoal **by** (*auto simp*: *term-poly-list-rel-Cons-iff rel2p-def term-poly-list-rel-remdups-mset*)

subgoal **for** $x \ p \ p' \ q'$

by (*auto simp*: *term-poly-list-rel-Cons-iff rel2p-def term-poly-list-rel-remdups-mset*

dest!: *multi-member-split*[*of* - q'])

subgoal **premises** p **for** $x \ p \ y \ q \ p' \ q'$

apply (*cases* $\langle x = y \rangle$)

subgoal

using $p(1)$ [*of* $\langle \text{remove1-mset } y \ p' \rangle \langle \text{remove1-mset } y \ q' \rangle$] $p(4-)$

by (*auto simp*: *term-poly-list-rel-Cons-iff rel2p-def*

dest!: *var-notin-notin-mult-monomsD*

dest!: *multi-member-split*)

apply (*cases* $\langle (x, y) \in \text{var-order-rel} \rangle$)

subgoal

using $p(2)$ [*of* $\langle \text{remove1-mset } x \ p' \rangle \langle q' \rangle$] $p(4-)$

apply (*auto simp*: *term-poly-list-rel-Cons-iff*

term-poly-list-rel-set-mset rel2p-def var-order-rel-def

dest!: *multi-member-split*[*of* - p'] *multi-member-split*[*of* - q']

var-notin-notin-mult-monomsD

split: *if-splits*)

apply (*meson* *lexord-cons-cons list.inject total-on-lexord-less-than-char-linear*)

apply (*meson* *lexord-cons-cons list.inject total-on-lexord-less-than-char-linear*)

apply (*meson* *lexord-cons-cons list.inject total-on-lexord-less-than-char-linear*)

using *lexord-trans trans-less-than-char var-order-rel-antisym*

unfolding *var-order-rel-def* **apply** *blast+*

done

subgoal

using $p(3)$ [*of* $\langle p' \rangle \langle \text{remove1-mset } y \ q' \rangle$] $p(4-)$

apply (*auto simp*: *term-poly-list-rel-Cons-iff rel2p-def*

term-poly-list-rel-set-mset rel2p-def var-order-rel-antisym

dest!: *multi-member-split*[*of* - p'] *multi-member-split*[*of* - q']

var-notin-notin-mult-monomsD

split: *if-splits*)

using *lexord-trans trans-less-than-char var-order-rel-antisym*

unfolding *var-order-rel-def* **apply** *blast*

```

    apply (meson lexord-cons-cons list.inject total-on-lexord-less-than-char-linear)
    by (meson less-than-char-linear lexord-linear lexord-trans trans-less-than-char)
  done
done
done
qed

```

definition *mult-monomials* :: $\langle \text{term-poly-list} \times \text{int} \Rightarrow \text{term-poly-list} \times \text{int} \Rightarrow \text{term-poly-list} \times \text{int} \rangle$
where

```

  ⟨mult-monomials = (λ(x, a) (y, b). (mult-monom x y, a * b))⟩

```

definition *mult-poly-raw* :: $\langle \text{l-list-polynomial} \Rightarrow \text{l-list-polynomial} \Rightarrow \text{l-list-polynomial} \rangle$ **where**

```

  ⟨mult-poly-raw p q = foldl (λb x. map (mult-monomials x) q @ b) [] p⟩

```

fun *map-append* **where**

```

  ⟨map-append f b [] = b⟩ |
  ⟨map-append f b (x # xs) = f x # map-append f b xs⟩

```

lemma *map-append-alt-def*:

```

  ⟨map-append f b xs = map f xs @ b⟩
by (induction f b xs rule: map-append.induct)
  auto

```

lemma *foldl-append-empty*:

```

  ⟨NO-MATCH [] xs  $\implies$  foldl (λb x. f x @ b) xs p = foldl (λb x. f x @ b) [] p @ xs⟩
apply (induction p arbitrary: xs)
apply simp
by (metis (mono-tags, lifting) NO-MATCH-def append.assoc append-self-conv foldl-Cons)

```

lemma *poly-list-rel-empty-iff[simp]*:

```

  ⟨([], r) ∈ poly-list-rel R  $\iff$  r = {#}⟩
by (auto simp: poly-list-rel-def list-mset-rel-def br-def)

```

lemma *mult-poly-raw-simp[simp]*:

```

  ⟨mult-poly-raw [] q = []⟩
  ⟨mult-poly-raw (x # p) q = mult-poly-raw p q @ map (mult-monomials x) q⟩
subgoal by (auto simp: mult-poly-raw-def)
subgoal by (induction p) (auto simp: mult-poly-raw-def foldl-append-empty)
done

```

lemma *sorted-poly-list-relD*:

```

  ⟨(q, q') ∈ sorted-poly-list-rel R  $\implies$  q' = (λ(a, b). (mset a, b)) '# mset q⟩
apply (induction q arbitrary: q')
apply (auto simp: sorted-poly-list-rel-wrt-def list-mset-rel-def br-def
  list-rel-split-right-iff)
apply (subst (asm)(2) term-poly-list-rel-def)
apply (simp add: relcomp.relcompI)
done

```

lemma *list-all2-in-set-ExD*:

```

  ⟨list-all2 R p q  $\implies$  x ∈ set p  $\implies$  ∃ y ∈ set q. R x y⟩
by (induction p q rule: list-all2-induct)
  auto

```

inductive-cases *mult-poly-p-elim*: $\langle \text{mult-poly-p } q \text{ (A, r) (B, r')} \rangle$

lemma *mult-poly-p-add-mset-same*:

$\langle (\text{mult-poly-p } q)^{**} \text{ (A, r) (B, r')} \implies (\text{mult-poly-p } q)^{**} \text{ (add-mset } x \text{ A, r) (add-mset } x \text{ B, r')} \rangle$
apply (*induction rule*: *rtranclp-induct*[of $\langle \text{mult-poly-p } q \rangle \langle (-, r) \rangle \langle (p', q') \rangle$ **for** $p' \ q''$, *split-format*(*complete*)])
subgoal by *simp*
apply (*rule* *rtranclp.rtrancl-into-rtrancl*)
apply *assumption*
by (*auto elim!*: *mult-poly-p-elim intro: mult-poly-p.intros*
intro: rtranclp.rtrancl-into-rtrancl simp: add-mset-commute[of x])

lemma *mult-poly-raw-mult-poly-p*:

assumes $\langle (p, p') \in \text{sorted-poly-rel} \rangle$ **and** $\langle (q, q') \in \text{sorted-poly-rel} \rangle$
shows $\langle \exists r. (\text{mult-poly-raw } p \ q, r) \in \text{unsorted-poly-rel} \wedge (\text{mult-poly-p } q')^{**} \text{ (p', \{\#\}) (\{\#\}, r) \rangle$

proof –

have H : $\langle (q, q') \in \text{sorted-poly-list-rel term-order} \implies n < \text{length } q \implies$
distinct aa $\implies \text{sorted-wrt var-order } aa \implies$
 $(\text{mult-monoms } aa \text{ (fst } (q \ ! \ n)),$
 $\text{mset } (\text{mult-monoms } aa \text{ (fst } (q \ ! \ n))))$
 $\in \text{term-poly-list-rel} \rangle$ **for** $aa \ n$
using *mult-monoms-spec*[*unfolded fun-rel-def, simplified*] **apply** –
apply (*drule bspec*[of - - $\langle (aa, (\text{mset } aa)) \rangle$])
apply (*auto simp: term-poly-list-rel-def*)[]
unfolding *prod.case sorted-poly-list-rel-wrt-def*
apply *clarsimp*
subgoal for y
apply (*drule bspec*[of - - $\langle (\text{fst } (q \ ! \ n), \text{mset } (\text{fst } (q \ ! \ n))) \rangle$])
apply (*cases* $\langle q \ ! \ n \rangle$; *cases* $\langle y \ ! \ n \rangle$)
using *param-nth*[of $n \ y \ n \ q \ \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle$]
by (*auto simp: list-rel-imp-same-length term-poly-list-rel-def*)
done

have H' : $\langle (q, q') \in \text{sorted-poly-list-rel term-order} \implies$
distinct aa $\implies \text{sorted-wrt var-order } aa \implies$
 $(ab, ba) \in \text{set } q \implies$
 $\text{remdups-mset } (\text{mset } aa + \text{mset } ab) = \text{mset } (\text{mult-monoms } aa \ ab) \rangle$ **for** $aa \ n \ ab \ ba$
using *mult-monoms-spec*[*unfolded fun-rel-def, simplified*] **apply** –
apply (*drule bspec*[of - - $\langle (aa, (\text{mset } aa)) \rangle$])
apply (*auto simp: term-poly-list-rel-def*)[]
unfolding *prod.case sorted-poly-list-rel-wrt-def*
apply *clarsimp*
subgoal for y
apply (*drule bspec*[of - - $\langle (ab, \text{mset } ab) \rangle$])
apply (*auto simp: list-rel-imp-same-length term-poly-list-rel-def list-rel-def*
dest: list-all2-in-set-ExD)
done
done

have H : $\langle (q, q') \in \text{sorted-poly-list-rel term-order} \implies$
 $a = (aa, b) \implies$
 $(pq, r) \in \text{unsorted-poly-rel} \implies$
 $p' = \text{add-mset } (\text{mset } aa, b) \ A \implies$
 $\forall x \in \text{set } p. \text{term-order } aa \text{ (fst } x) \implies$
 $\text{sorted-wrt var-order } aa \implies$

```

distinct aa  $\implies$  b  $\neq$  0  $\implies$ 
( $\bigwedge$ aaa. (aaa, 0)  $\notin$  # q')  $\implies$ 
(pq @
  map (mult-monomials (aa, b)) q,
  {#case x of (ys, n)  $\Rightarrow$  (remdups-mset (mset aa + ys), n * b)
  . x  $\in$  # q' #} +
  r)
 $\in$  unsorted-poly-rel for a p p' pq aa b r
apply (auto simp: poly-list-rel-def)
apply (rule-tac b =  $\langle$ y @ map ( $\lambda$ (a,b). (mset a, b)) (map (mult-monomials (aa, b)) q) $\rangle$  in relcompI)
apply (auto simp: list-rel-def list-all2-append list-all2-lengthD H
  list-mset-rel-def br-def mult-monomials-def case-prod-beta intro!: list-all2-all-nthI
  simp: sorted-poly-list-relD)
apply (subst sorted-poly-list-relD[of q q' term-order])
apply (auto simp: case-prod-beta H' intro!: image-mset-cong)
done

```

```

show ?thesis
using assms
apply (induction p arbitrary: p')
subgoal
  by auto
subgoal premises p for a p p'
  using p(1)[of  $\langle$ remove1-mset (mset (fst a), snd a) p' $\rangle$ ] p(2-)
  apply (cases a)
  apply (auto simp: sorted-poly-list-rel-Cons-iff
    dest!: multi-member-split)
  apply (rule-tac x =  $\langle$ ( $\lambda$ (ys, n). (remdups-mset (mset (fst a) + ys), n * snd a))  $\notin$  # q' + r $\rangle$  in exI)
  apply (auto 5 3 intro: mult-poly-p.intros simp: intro!: H
    dest: sorted-poly-list-rel-nonzeroD nonzero-coeffsD)
  apply (rule rtranclp-trans)
  apply (rule mult-poly-p-add-mset-same)
  apply assumption
  apply (rule converse-rtranclp-into-rtranclp)
  apply (auto intro!: mult-poly-p.intros simp: ac-simps)
  done
done
qed

```

```

fun merge-coeffs ::  $\langle$ llist-polynomial  $\Rightarrow$  llist-polynomial $\rangle$  where
   $\langle$ merge-coeffs [] = [] $\rangle$  |
   $\langle$ merge-coeffs [(xs, n)] = [(xs, n)] $\rangle$  |
   $\langle$ merge-coeffs ((xs, n) # (ys, m) # p) =
    (if xs = ys
     then if n + m  $\neq$  0 then merge-coeffs ((xs, n + m) # p) else merge-coeffs p
     else (xs, n) # merge-coeffs ((ys, m) # p)) $\rangle$ 

```

```

abbreviation (in -)monoms ::  $\langle$ llist-polynomial  $\Rightarrow$  term-poly-list set $\rangle$  where
   $\langle$ monoms p  $\equiv$  fst 'set p $\rangle$ 

```

```

lemma fst-normalize-polynomial-subset:
   $\langle$ monoms (merge-coeffs p)  $\subseteq$  monoms p $\rangle$ 
by (induction p rule: merge-coeffs.induct) auto

```

lemma *fst-normalize-polynomial-subsetD*:

$\langle (a, b) \in \text{set } (\text{merge-coeffs } p) \implies a \in \text{monoms } p \rangle$

apply (*induction* *p* *rule*: *merge-coeffs.induct*)

subgoal

by *auto*

subgoal

by *auto*

subgoal

by (*auto split*: *if-splits*)

done

lemma *distinct-merge-coeffs*:

assumes $\langle \text{sorted-wrt } R (\text{map } \text{fst } xs) \rangle$ **and** $\langle \text{transp } R \rangle$ $\langle \text{antisymp } R \rangle$

shows $\langle \text{distinct } (\text{map } \text{fst } (\text{merge-coeffs } xs)) \rangle$

using *assms*

by (*induction* *xs* *rule*: *merge-coeffs.induct*)

(*auto* 5 4 *dest*: *antisympD* *dest!*: *fst-normalize-polynomial-subsetD*)

lemma *in-set-merge-coeffsD*:

$\langle (a, b) \in \text{set } (\text{merge-coeffs } p) \implies \exists b. (a, b) \in \text{set } p \rangle$

by (*auto* *dest!*: *fst-normalize-polynomial-subsetD*)

lemma *rtranclp-normalize-poly-add-mset*:

$\langle \text{normalize-poly-p}^{**} A r \implies \text{normalize-poly-p}^{**} (\text{add-mset } x A) (\text{add-mset } x r) \rangle$

by (*induction* *rule*: *rtranclp-induct*)

(*auto* *dest*: *normalize-poly-p.keep-coeff*[*of* - - *x*])

lemma *nonzero-coeffs-diff*:

$\langle \text{nonzero-coeffs } A \implies \text{nonzero-coeffs } (A - B) \rangle$

by (*auto simp*: *nonzero-coeffs-def* *dest*: *in-diffD*)

lemma *merge-coeffs-is-normalize-poly-p*:

$\langle (xs, ys) \in \text{sorted-repeat-poly-rel} \implies \exists r. (\text{merge-coeffs } xs, r) \in \text{sorted-poly-rel} \wedge \text{normalize-poly-p}^{**} ys r \rangle$

apply (*induction* *xs* *arbitrary*: *ys* *rule*: *merge-coeffs.induct*)

subgoal **by** (*auto simp*: *sorted-repeat-poly-list-rel-wrt-def* *sorted-poly-list-rel-wrt-def*)

subgoal

by (*auto simp*: *sorted-repeat-poly-list-rel-wrt-def* *sorted-poly-list-rel-wrt-def*)

subgoal **premises** *p* **for** *xs n ys m p ysa*

apply (*cases* $\langle xs = ys \rangle$, *cases* $\langle m+n \neq 0 \rangle$)

subgoal

using $p(1)[\text{of } \langle \text{add-mset } (\text{mset } ys, m+n) ysa - \{\#(\text{mset } ys, m), (\text{mset } ys, n)\#} \rangle] p(4-)$

apply (*auto simp*: *sorted-poly-list-rel-Cons-iff* *ac-simps* *add-mset-commute*

remove1-mset-add-mset-If *nonzero-coeffs-diff* *sorted-repeat-poly-list-rel-Cons-iff*)

apply (*rule-tac* *x* = $\langle r \rangle$ **in** *exI*)

using $\text{normalize-poly-p.merge-dup-coeff}[\text{of } \langle ysa - \{\#(\text{mset } ys, m), (\text{mset } ys, n)\#} \rangle \langle ysa - \{\#(\text{mset } ys, m), (\text{mset } ys, n)\#} \rangle \langle \text{mset } ys \rangle m n]$

by (*auto* *dest!*: *multi-member-split* *simp* *del*: *normalize-poly-p.merge-dup-coeff*

simp: *add-mset-commute*

intro: *converse-rtranclp-into-rtranclp*)

subgoal

using $p(2)[\text{of } \langle ysa - \{\#(\text{mset } ys, m), (\text{mset } ys, n)\#} \rangle] p(4-)$

apply (*auto simp*: *sorted-poly-list-rel-Cons-iff* *ac-simps* *add-mset-commute*)


```

    remove1-mset-add-mset-If nonzero-coeffs-diff sorted-repeat-poly-list-rel-Cons-iff)
apply (rule-tac x = ⟨r⟩ in exI)
    using normalize-poly-p.rem-0-coeff[of ⟨add-mset (mset ys, m + n) ysa - {#(mset ys, m), (mset
ys, n)#}⟩ ⟨add-mset (mset ys, m + n) ysa - {#(mset ys, m), (mset ys, n)#}⟩ ⟨mset ys⟩]
    using normalize-poly-p.merge-dup-coeff[of ⟨ysa - {#(mset ys, m), (mset ys, n)#}⟩ ⟨ysa -
{#(mset ys, m), (mset ys, n)#}⟩ ⟨mset ys⟩ m n]
    by (force intro: add-mset-commute[of ⟨(mset ys, n)⟩ ⟨(mset ys, -n)⟩]
    converse-rtranclp-into-rtranclp
    dest!: multi-member-split
    simp del: normalize-poly-p.rem-0-coeff
    simp: add-eq-0-iff2
    intro: normalize-poly-p.rem-0-coeff)
subgoal
    using p(3)[of ⟨add-mset (mset ys, m) ysa - {#(mset xs, n), (mset ys, m)#}⟩] p(4-)
apply (auto simp: sorted-poly-list-rel-Cons-iff ac-simps add-mset-commute
    remove1-mset-add-mset-If sorted-repeat-poly-list-rel-Cons-iff)
apply (rule-tac x = ⟨add-mset (mset xs, n) r⟩ in exI)
apply (auto dest!: in-set-merge-coeffsD)
apply (auto intro: normalize-poly-p.intros rtranclp-normalize-poly-add-mset
    simp: rel2p-def var-order-rel-def
    dest!: multi-member-split
    dest: sorted-poly-list-rel-nonzeroD)
    using total-on-lexord-less-than-char-linear apply fastforce
    using total-on-lexord-less-than-char-linear apply fastforce
done
done
done

```

8.5 Normalisation

definition *normalize-poly* **where**

```

⟨normalize-poly p = do {
  p ← sort-poly-spec p;
  RETURN (merge-coeffs p)
}⟩

```

definition *sort-coeff* :: ⟨string list ⇒ string list nres⟩ **where**

```

⟨sort-coeff ys = SPEC(λxs. mset xs = mset ys ∧ sorted-wrt (rel2p (Id ∪ var-order-rel)) xs)⟩

```

lemma *distinct-var-order-Id-var-order*:

```

⟨distinct a ⇒ sorted-wrt (rel2p (Id ∪ var-order-rel)) a ⇒
  sorted-wrt var-order a⟩
by (induction a) (auto simp: rel2p-def)

```

definition *sort-all-coeffs* :: ⟨llist-polynomial ⇒ llist-polynomial nres⟩ **where**

```

⟨sort-all-coeffs xs = monadic-nfoldli xs (λ-. RETURN True) (λ(a, n) b. do {a ← sort-coeff a; RETURN
((a, n) # b)}) []⟩

```

lemma *sort-all-coeffs-gen*:

```

assumes ⟨(∀ xs ∈ monoms xs'. sorted-wrt (rel2p (var-order-rel)) xs)⟩ and
  ⟨∀ x ∈ monoms (xs @ xs'). distinct x⟩
shows ⟨monadic-nfoldli xs (λ-. RETURN True) (λ(a, n) b. do {a ← sort-coeff a; RETURN ((a, n)
# b)}) xs' ≤
  ↓Id (SPEC(λys. map (λ(a,b). (mset a, b)) (rev xs @ xs') = map (λ(a,b). (mset a, b)) (ys) ∧
  (∀ xs ∈ monoms ys. sorted-wrt (rel2p (var-order-rel)) xs)))⟩

```

proof –

```

have H: ⟨

```

```

  ∀ x ∈ set xs'. sorted-wrt var-order (fst x) ⇒
  sorted-wrt (rel2p (Id ∪ var-order-rel)) x ⇒
  (aaa, ba) ∈ set xs' ⇒
  sorted-wrt (rel2p (Id ∪ var-order-rel)) aaa for xs xs' ba aa b x aaa
by (metis UnCI fst-eqD rel2p-def sorted-wrt-mono-rel)
show ?thesis
using assms
unfolding sort-all-coeffs-def sort-coeff-def
apply (induction xs arbitrary: xs')
subgoal
  using assms
  by auto
subgoal premises p for a xs
  using p(2-)
apply (cases a, simp only: monadic-nfoldli-simp bind-to-let-conv Let-def if-True Refine-Basic.nres-monad3
  intro-spec-refine-iff prod.case)
  by (auto 5 3 simp: intro-spec-refine-iff image-Un
  dest: same-mset-distinct-iff
  intro!: p(1)[THEN order-trans] distinct-var-order-Id-var-order
  simp: H)
done
qed

```

definition *shuffle-coefficients* **where**

```

⟨shuffle-coefficients xs = (SPEC(λys. map (λ(a,b). (mset a, b)) (rev xs) = map (λ(a,b). (mset a, b))
ys ∧
  (∀ xs ∈ mononoms ys. sorted-wrt (rel2p (var-order-rel)) xs)))⟩

```

lemma *sort-all-coeffs*:

```

⟨∀ x ∈ mononoms xs. distinct x ⇒
sort-all-coeffs xs ≤ ↓ Id (shuffle-coefficients xs)⟩
unfolding sort-all-coeffs-def shuffle-coefficients-def
by (rule sort-all-coeffs-gen[THEN order-trans])
  auto

```

lemma *unsorted-term-poly-list-rel-mset*:

```

⟨(ys, aa) ∈ unsorted-term-poly-list-rel ⇒ mset ys = aa⟩
by (auto simp: unsorted-term-poly-list-rel-def)

```

lemma *RETURN-map-alt-def*:

```

⟨RETURN o (map f) =
  RECT (λg xs.
    case xs of
      [] ⇒ RETURN []
    | x # xs ⇒ do {xs ← g xs; RETURN (f x # xs)}⟩

```

```

unfolding comp-def
apply (subst eq-commute)
apply (intro ext)
apply (induct-tac x)
subgoal
  apply (subst RECT-unfold)
  apply refine-mono
  apply auto
done
subgoal

```

```

apply (subst RECT-unfold)
apply refine-mono
apply auto
done
done

```

lemma *fully-unsorted-poly-rel-Cons-iff*:

```

⟨((ys, n) # p, a) ∈ fully-unsorted-poly-rel ⟷
  (p, remove1-mset (mset ys, n) a) ∈ fully-unsorted-poly-rel ∧
  (mset ys, n) ∈# a ∧ distinct ys⟩

```

```

apply (auto simp: poly-list-rel-def list-rel-split-right-iff list-mset-rel-def br-def
  unsorted-term-poly-list-rel-def

```

```

  nonzero-coeffs-def fully-unsorted-poly-list-rel-def dest!: multi-member-split)

```

```

apply blast

```

```

apply (rule-tac b = ⟨(mset ys, n) # y⟩ in relcompI)

```

```

apply auto

```

```

done

```

lemma *map-mset-unsorted-term-poly-list-rel*:

```

⟨(∧ a. a ∈ monoms s ⟹ distinct a) ⟹ ∀ x ∈ monoms s. distinct x ⟹
  (∀ xs ∈ monoms s. sorted-wrt (rel2p (Id ∪ var-order-rel)) xs) ⟹
  (s, map (λ(a, y). (mset a, y)) s)
  ∈ ⟨term-poly-list-rel ×r int-rel⟩list-rel⟩

```

```

by (induction s) (auto simp: term-poly-list-rel-def
  distinct-var-order-Id-var-order)

```

lemma *list-rel-unsorted-term-poly-list-relD*:

```

⟨(p, y) ∈ ⟨unsorted-term-poly-list-rel ×r int-rel⟩list-rel ⟹
  mset y = (λ(a, y). (mset a, y)) '# mset p ∧ (∀ x ∈ monoms p. distinct x)⟩

```

```

by (induction p arbitrary: y)

```

```

  (auto simp: list-rel-split-right-iff

```

```

  unsorted-term-poly-list-rel-def)

```

lemma *shuffle-terms-distinct-iff*:

```

assumes ⟨map (λ(a, y). (mset a, y)) p = map (λ(a, y). (mset a, y)) s⟩

```

```

shows ⟨(∀ x ∈ set p. distinct (fst x)) ⟷ (∀ x ∈ set s. distinct (fst x))⟩

```

proof –

```

have ⟨∀ x ∈ set s. distinct (fst x)⟩

```

```

if m: ⟨map (λ(a, y). (mset a, y)) p = map (λ(a, y). (mset a, y)) s⟩ and

```

```

  dist: ⟨∀ x ∈ set p. distinct (fst x)⟩

```

```

for s p

```

proof *standard+*

```

fix x

```

```

assume x: ⟨x ∈ set s⟩

```

```

obtain v n where [simp]: ⟨x = (v, n)⟩ by (cases x)

```

```

then have ⟨(mset v, n) ∈ set (map (λ(a, y). (mset a, y)) p)⟩

```

```

  using x unfolding m by auto

```

```

then obtain v' where

```

```

  ⟨(v', n) ∈ set p⟩ and

```

```

  ⟨mset v' = mset v⟩

```

```

by (auto simp: image-iff)

```

```

then show ⟨distinct (fst x)⟩

```

```

  using dist by (metis ⟨x = (v, n)⟩ distinct-mset-mset-distinct fst-conv)

```

qed

from *this*[of p s] *this*[of s p]
show $\langle ?thesis \rangle$
unfolding *assms*
by *blast*
qed

lemma

$\langle (p, y) \in \langle \text{unsorted-term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$
 $(a, b) \in \text{set } p \implies \text{distinct } a \rangle$
using *list-rel-unsorted-term-poly-list-relD* **by** *fastforce*

lemma *sort-all-coeffs-unsorted-poly-rel-with0*:

assumes $\langle (p, p') \in \text{fully-unsorted-poly-rel} \rangle$
shows $\langle \text{sort-all-coeffs } p \leq \Downarrow (\text{unsorted-poly-rel-with0}) (\text{RETURN } p') \rangle$

proof –

have H : $\langle \text{map } (\lambda(a, y). (\text{mset } a, y)) (\text{rev } p) =$
 $\text{map } (\lambda(a, y). (\text{mset } a, y)) s \longleftrightarrow$
 $(\text{map } (\lambda(a, y). (\text{mset } a, y)) p) =$
 $\text{map } (\lambda(a, y). (\text{mset } a, y)) (\text{rev } s) \rangle$ **for** s
by (*auto simp flip: rev-map simp: eq-commute*[of $\langle \text{rev } (\text{map } - -) \rangle \langle \text{map } - - \rangle$])
have 1 : $\langle \bigwedge s y. (p, y) \in \langle \text{unsorted-term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$
 $p' = \text{mset } y \implies$
 $\text{map } (\lambda(a, y). (\text{mset } a, y)) (\text{rev } p) = \text{map } (\lambda(a, y). (\text{mset } a, y)) s \implies$
 $\forall x \in \text{set } s. \text{sorted-wrt var-order } (\text{fst } x) \implies$
 $(s, \text{map } (\lambda(a, y). (\text{mset } a, y)) s)$
 $\in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \rangle$
by (*auto 4 4 simp: rel2p-def*
dest!: list-rel-unsorted-term-poly-list-relD
dest: shuffle-terms-distinct-iff[*THEN iffD1*]
intro!: map-mset-unsorted-term-poly-list-rel
sorted-wrt-mono-rel[of $- \langle \text{rel2p } (\text{var-order-rel}) \rangle \langle \text{rel2p } (\text{Id} \cup \text{var-order-rel}) \rangle$])
have 2 : $\langle \bigwedge s y. (p, y) \in \langle \text{unsorted-term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$
 $p' = \text{mset } y \implies$
 $\text{map } (\lambda(a, y). (\text{mset } a, y)) (\text{rev } p) = \text{map } (\lambda(a, y). (\text{mset } a, y)) s \implies$
 $\forall x \in \text{set } s. \text{sorted-wrt var-order } (\text{fst } x) \implies$
 $\text{mset } y = \{ \# \text{case } x \text{ of } (a, x) \Rightarrow (\text{mset } a, x). x \in \# \text{mset } s \# \} \rangle$
by (*metis* (*no-types, lifting*) *list-rel-unsorted-term-poly-list-relD mset-map mset-rev*)
show *?thesis*
apply (*rule sort-all-coeffs*[*THEN order-trans*])
using *assms*
by (*auto simp: shuffle-coefficients-def poly-list-rel-def*
RETURN-def fully-unsorted-poly-list-rel-def list-mset-rel-def
br-def dest: list-rel-unsorted-term-poly-list-relD
intro!: RES-refine relcompI[of $- \langle \text{map } (\lambda(a, y). (\text{mset } a, y)) (\text{rev } p) \rangle$]
 1 2)

qed

lemma *sort-poly-spec-id'*:

assumes $\langle (p, p') \in \text{unsorted-poly-rel-with0} \rangle$
shows $\langle \text{sort-poly-spec } p \leq \Downarrow (\text{sorted-repeat-poly-rel-with0}) (\text{RETURN } p') \rangle$

proof –

obtain y **where**
 py : $\langle (p, y) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \rangle$ **and**
 $p'-y$: $\langle p' = \text{mset } y \rangle$
using *assms*

```

unfolding fully-unsorted-poly-list-rel-def poly-list-rel-def sorted-poly-list-rel-wrt-def
  by (auto simp: list-mset-rel-def br-def)
then have [simp]:  $\langle \text{length } y = \text{length } p \rangle$ 
  by (auto simp: list-rel-def list-all2-conv-all-nth)
have  $H: \langle (x, p') \in \langle \text{term-poly-list-rel } \times_r \text{ int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \rangle$ 
  if  $px: \langle \text{mset } p = \text{mset } x \rangle$  and  $\langle \text{sorted-wrt } (\text{rel2p } (Id \cup \text{lexord var-order-rel})) (\text{map fst } x) \rangle$ 
  for  $x :: \langle \text{llist-polynomial} \rangle$ 
proof –
  from  $px$  have  $\langle \text{length } x = \text{length } p \rangle$ 
    by (metis size-mset)
  from  $px$  have  $\langle \text{mset } x = \text{mset } p \rangle$ 
    by simp
  then obtain  $f$  where  $\langle f \text{ permutes } \{..<\text{length } p\} \rangle \langle \text{permute-list } f p = x \rangle$ 
    by (rule mset-eq-permutation)
  with  $\langle \text{length } x = \text{length } p \rangle$  have  $f: \langle \text{bij-betw } f \{..<\text{length } x\} \{..<\text{length } p\} \rangle$ 
    by (simp add: permutes-imp-bij)
  from  $\langle f \text{ permutes } \{..<\text{length } p\} \rangle \langle \text{permute-list } f p = x \rangle$  [symmetric]
  have [simp]:  $\langle \bigwedge i. i < \text{length } x \implies x ! i = p ! (f i) \rangle$ 
    by (simp add: permute-list-nth)
  let  $?y = \langle \text{map } (\lambda i. y ! f i) [0 ..< \text{length } x] \rangle$ 
  have  $\langle i < \text{length } y \implies (p ! f i, y ! f i) \in \text{term-poly-list-rel } \times_r \text{ int-rel} \rangle$  for  $i$ 
    using list-all2-nthD[of - p y]
       $\langle f i \rangle, OF$  py[unfolded list-rel-def mem-Collect-eq prod.case]
      mset-eq-length[OF px]  $f$ 
    by (auto simp: list-rel-def list-all2-conv-all-nth bij-betw-def)
  then have  $\langle (x, ?y) \in \langle \text{term-poly-list-rel } \times_r \text{ int-rel} \rangle \text{list-rel} \rangle$  and
     $xy: \langle \text{length } x = \text{length } y \rangle$ 
    using py list-all2-nthD[of  $\langle \text{rel2p } (\text{term-poly-list-rel } \times_r \text{ int-rel}) \rangle p y$ ]
       $\langle f i \rangle$  for  $i$ , simplified] mset-eq-length[OF px]
    by (auto simp: list-rel-def list-all2-conv-all-nth)
  moreover {
    have  $f: \langle \text{mset-set } \{0..<\text{length } x\} = f \# \text{mset-set } \{0..<\text{length } x\} \rangle$ 
      using f mset-eq-length[OF px]
      by (auto simp: bij-betw-def lessThan-atLeast0 image-mset-mset-set)
    have  $\langle \text{mset } y = \{ \#y ! f x. x \in \# \text{mset-set } \{0..<\text{length } x\} \# \} \rangle$ 
      by (subst drop-0[symmetric], subst mset-drop-upto, subst xy[symmetric], subst f)
      auto
    then have  $\langle (?y, p') \in \text{list-mset-rel} \rangle$ 
      by (auto simp: list-mset-rel-def br-def p'-y)
  }
  ultimately show ?thesis
    by (auto intro!: relcompI[of - ?y])
qed
show ?thesis
  unfolding sort-poly-spec-def poly-list-rel-def sorted-repeat-poly-list-rel-with0-wrt-def
  by refine-rcg (auto intro: H)
qed

```

```

fun merge-coeffs0 ::  $\langle \text{llist-polynomial} \Rightarrow \text{llist-polynomial} \rangle$  where
   $\langle \text{merge-coeffs0 } [] = [] \rangle |$ 
   $\langle \text{merge-coeffs0 } [(x, n)] = (\text{if } n = 0 \text{ then } [] \text{ else } [(x, n)]) \rangle |$ 
   $\langle \text{merge-coeffs0 } ((x, n) \# (y, m) \# p) =$ 
     $(\text{if } xs = ys$ 

```

then if $n + m \neq 0$ then $\text{merge-coeffs0 } ((xs, n + m) \# p)$ else $\text{merge-coeffs0 } p$
 else if $n = 0$ then $\text{merge-coeffs0 } ((ys, m) \# p)$
 else $(xs, n) \# \text{merge-coeffs0 } ((ys, m) \# p)$

lemma *sorted-repeat-poly-list-rel-with0-wrt-ConsD:*

$\langle ((ys, n) \# p, a) \in \text{sorted-repeat-poly-list-rel-with0-wrt } S \text{ term-poly-list-rel} \implies$
 $(p, \text{remove1-mset } (\text{mset } ys, n) a) \in \text{sorted-repeat-poly-list-rel-with0-wrt } S \text{ term-poly-list-rel} \wedge$
 $(\text{mset } ys, n) \in \# a \wedge (\forall x \in \text{set } p. S \text{ } ys \text{ } (\text{fst } x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } ys \wedge$
 $\text{distinct } ys \rangle$

unfolding *sorted-repeat-poly-list-rel-with0-wrt-def prod.case mem-Collect-eq list-rel-def*

apply (*clarsimp*)
apply (*subst (asm) list.rel.sel*)
apply (*intro conjI*)
apply (*rule-tac b = <tl y> in relcompI*)
apply (*auto simp: sorted-poly-list-rel-wrt-def list-mset-rel-def br-def*)
apply (*case-tac <lead-coeff y>; case-tac y*)
apply (*auto simp: term-poly-list-rel-def*)
apply (*case-tac <lead-coeff y>; case-tac y*)
apply (*auto simp: term-poly-list-rel-def*)
apply (*case-tac <lead-coeff y>; case-tac y*)
apply (*auto simp: term-poly-list-rel-def*)
apply (*case-tac <lead-coeff y>; case-tac y*)
apply (*auto simp: term-poly-list-rel-def*)
done

lemma *sorted-repeat-poly-list-rel-with0-wrtl-Cons-iff:*

$\langle ((ys, n) \# p, a) \in \text{sorted-repeat-poly-list-rel-with0-wrt } S \text{ term-poly-list-rel} \iff$
 $(p, \text{remove1-mset } (\text{mset } ys, n) a) \in \text{sorted-repeat-poly-list-rel-with0-wrt } S \text{ term-poly-list-rel} \wedge$
 $(\text{mset } ys, n) \in \# a \wedge (\forall x \in \text{set } p. S \text{ } ys \text{ } (\text{fst } x)) \wedge \text{sorted-wrt } (\text{rel2p var-order-rel}) \text{ } ys \wedge$
 $\text{distinct } ys \rangle$

apply (*rule iffI*)

subgoal

by (*auto dest!: sorted-repeat-poly-list-rel-with0-wrt-ConsD*)

subgoal

unfolding *sorted-poly-list-rel-wrt-def prod.case mem-Collect-eq list-rel-def sorted-repeat-poly-list-rel-with0-wrt-def*

apply (*clarsimp*)

apply (*rule-tac b = <(mset ys, n) \# y> in relcompI*)

by (*auto simp: sorted-poly-list-rel-wrt-def list-mset-rel-def br-def*
term-poly-list-rel-def add-mset-eq-add-mset eq-commute[of - <mset ->]
nonzero-coeffs-def
dest!: multi-member-split)

done

lemma *fst-normalize0-polynomial-subsetD:*

$\langle (a, b) \in \text{set } (\text{merge-coeffs0 } p) \implies a \in \text{monoms } p \rangle$

apply (*induction p rule: merge-coeffs0.induct*)

subgoal

by *auto*

subgoal

by (*auto split: if-splits*)

subgoal

by (*auto split: if-splits*)

done

lemma *in-set-merge-coeffs0D*:

$\langle (a, b) \in \text{set } (\text{merge-coeffs0 } p) \implies \exists b. (a, b) \in \text{set } p \rangle$
by (*auto dest!: fst-normalize0-polynomial-subsetD*)

lemma *merge-coeffs0-is-normalize-poly-p*:

$\langle (xs, ys) \in \text{sorted-repeat-poly-rel-with0} \implies \exists r. (\text{merge-coeffs0 } xs, r) \in \text{sorted-poly-rel} \wedge \text{normalize-poly-p}^{**} \text{ } ys \ r \rangle$

apply (*induction xs arbitrary: ys rule: merge-coeffs0.induct*)

subgoal by (*auto simp: sorted-repeat-poly-list-rel-wrt-def sorted-poly-list-rel-wrt-def sorted-repeat-poly-list-rel-with0-wrt-def list-mset-rel-def br-def*)

subgoal for *xs n ys*

by (*force simp: sorted-repeat-poly-list-rel-wrt-def sorted-poly-list-rel-wrt-def sorted-repeat-poly-list-rel-with0-wrt-def list-mset-rel-def br-def list-rel-split-right-iff*)

subgoal premises *p for xs n ys m p ysa*

apply (*cases <xs = ys>, cases <m+n ≠ 0>*)

subgoal

using *p(1)[of <add-mset (mset ys, m+n) ysa - {#(mset ys, m), (mset ys, n)#}>] p(5-)*

apply (*auto simp: sorted-repeat-poly-list-rel-with0-wrtl-Cons-iff ac-simps add-mset-commute remove1-mset-add-mset-If nonzero-coeffs-diff sorted-repeat-poly-list-rel-Cons-iff*)

apply (*auto intro: normalize-poly-p.intros add-mset-commute add-mset-commute converse-rtranclp-into-rtranclp dest!: multi-member-split simp del: normalize-poly-p.merge-dup-coeff*)

apply (*rule-tac x = <r> in exI*)

using *normalize-poly-p.merge-dup-coeff[of <ysa - {#(mset ys, m), (mset ys, n)#}> <ysa - {#(mset ys, m), (mset ys, n)#}> <mset ys> m n]*

by (*auto intro: normalize-poly-p.intros*

converse-rtranclp-into-rtranclp dest!: multi-member-split

simp: add-mset-commute[of <(mset ys, n)> <(mset ys, m)>]

simp del: normalize-poly-p.merge-dup-coeff)

subgoal

using *p(2)[of <ysa - {#(mset ys, m), (mset ys, n)#}>] p(5-)*

apply (*auto simp: sorted-repeat-poly-list-rel-with0-wrtl-Cons-iff ac-simps add-mset-commute remove1-mset-add-mset-If nonzero-coeffs-diff sorted-repeat-poly-list-rel-Cons-iff*)

apply (*rule-tac x = <r> in exI*)

using *normalize-poly-p.rem-0-coeff[of <add-mset (mset ys, m+n) ysa - {#(mset ys, m), (mset ys, n)#}> <add-mset (mset ys, m+n) ysa - {#(mset ys, m), (mset ys, n)#}> <mset ys>]*

using *normalize-poly-p.merge-dup-coeff[of <ysa - {#(mset ys, m), (mset ys, n)#}> <ysa - {#(mset ys, m), (mset ys, n)#}> <mset ys> m n]*

by (*force intro: normalize-poly-p.intros converse-rtranclp-into-rtranclp dest!: multi-member-split*

simp del: normalize-poly-p.rem-0-coeff

simp: add-mset-commute[of <(mset ys, n)> <(mset ys, m)>])

apply (*cases <n = 0>*)

subgoal

using *p(3)[of <add-mset (mset ys, m) ysa - {#(mset xs, n), (mset ys, m)#}>] p(4-)*

apply (*auto simp: sorted-repeat-poly-list-rel-with0-wrtl-Cons-iff ac-simps add-mset-commute remove1-mset-add-mset-If sorted-repeat-poly-list-rel-Cons-iff*)

apply (*rule-tac x = <r> in exI*)

apply (*auto dest!: in-set-merge-coeffsD*)

by (*force intro: rtranclp-normalize-poly-add-mset converse-rtranclp-into-rtranclp*

simp: rel2p-def var-order-rel-def sorted-poly-list-rel-Cons-iff)

```

    dest!: multi-member-split
    dest: sorted-poly-list-rel-nonzeroD)
subgoal
  using p(4)[of ⟨add-mset (mset ys, m) ysa - {#(mset xs, n), (mset ys, m)#}⟩] p(5-)
apply (auto simp: sorted-repeat-poly-list-rel-with0-wrtl-Cons-iff ac-simps add-mset-commute
  remove1-mset-add-mset-If sorted-repeat-poly-list-rel-Cons-iff)
apply (rule-tac x = ⟨add-mset (mset xs, n) r⟩ in exI)
apply (auto dest!: in-set-merge-coeffs0D)
apply (auto intro: normalize-poly-p.intros rtranclp-normalize-poly-add-mset
  simp: rel2p-def var-order-rel-def sorted-poly-list-rel-Cons-iff
  dest!: multi-member-split
  dest: sorted-poly-list-rel-nonzeroD)
using in-set-merge-coeffs0D total-on-lexord-less-than-char-linear apply fastforce
using in-set-merge-coeffs0D total-on-lexord-less-than-char-linear apply fastforce
done
done
done

```

definition *full-normalize-poly* **where**

```

⟨full-normalize-poly p = do {
  p ← sort-all-coeffs p;
  p ← sort-poly-spec p;
  RETURN (merge-coeffs0 p)
}⟩

```

fun *sorted-remdups* **where**

```

⟨sorted-remdups (x # y # zs) =
  (if x = y then sorted-remdups (y # zs) else x # sorted-remdups (y # zs))⟩ |
⟨sorted-remdups zs = zs⟩

```

lemma *set-sorted-remdups*[simp]:

```

⟨set (sorted-remdups xs) = set xs⟩
by (induction xs rule: sorted-remdups.induct)
  auto

```

lemma *distinct-sorted-remdups*:

```

⟨sorted-wrt R xs ⟹ transp R ⟹ Restricted-Predicates.total-on R UNIV ⟹
  antisymp R ⟹ distinct (sorted-remdups xs)⟩
by (induction xs rule: sorted-remdups.induct)
  (auto dest: antisympD)

```

lemma *full-normalize-poly-normalize-poly-p*:

```

assumes ⟨(p, p') ∈ fully-unsorted-poly-rel⟩
shows ⟨full-normalize-poly p ≤ ↓ (sorted-poly-rel) (SPEC (λr. normalize-poly-p** p' r))⟩
(is ⟨?A ≤ ↓ ?R ?B⟩)

```

proof –

```

have 1: ⟨?B = do {
  p' ← RETURN p';
  p' ← RETURN p';
  SPEC (λr. normalize-poly-p** p' r)
}⟩
by auto

```

```

have [refine0]: ⟨sort-all-coeffs p ≤ SPEC(λp. (p, p') ∈ unsorted-poly-rel-with0)⟩
by (rule sort-all-coeffs-unsorted-poly-rel-with0[OF assms, THEN order-trans])
  (auto simp: conc-fun-RES RETURN-def)

```



```

have [refine0]: ⟨sort-poly-spec p ≤ SPEC (λc. (c, p') ∈ sorted-repeat-poly-rel-with0)⟩
  if ⟨(p, p') ∈ unsorted-poly-rel-with0⟩
  for p p'
  by (rule sort-poly-spec-id'[THEN order-trans, OF that])
    (auto simp: conc-fun-RES RETURN-def)
show ?thesis
apply (subst 1)
unfolding full-normalize-poly-def
by (refine-rcg)
  (auto intro!: RES-refine
    dest!: merge-coeffs0-is-normalize-poly-p
    simp: RETURN-def)
qed

```

```

definition mult-poly-full :: ⟨-⟩ where
⟨mult-poly-full p q = do {
  let pq = mult-poly-raw p q;
  normalize-poly pq
}⟩

```

```

lemma normalize-poly-normalize-poly-p:
  assumes ⟨(p, p') ∈ unsorted-poly-rel⟩
  shows ⟨normalize-poly p ≤ ↓ (sorted-poly-rel) (SPEC (λr. normalize-poly-p** p' r))⟩
proof -
  have 1: ⟨SPEC (λr. normalize-poly-p** p' r) = do {
    p' ← RETURN p';
    SPEC (λr. normalize-poly-p** p' r)
  }⟩
  by auto
show ?thesis
  unfolding normalize-poly-def
  apply (subst 1)
  apply (refine-rcg sort-poly-spec-id[OF assms]
    merge-coeffs-is-normalize-poly-p)
  subgoal
  by (drule merge-coeffs-is-normalize-poly-p)
  (auto intro!: RES-refine simp: RETURN-def)
  done
qed

```

8.6 Multiplication and normalisation

```

definition mult-poly-p' :: ⟨-⟩ where
⟨mult-poly-p' p' q' = do {
  pq ← SPEC(λr. (mult-poly-p q')** (p', {#}) ({#}, r));
  SPEC (λr. normalize-poly-p** pq r)
}⟩

```

```

lemma unsorted-poly-rel-fully-unsorted-poly-rel:
  ⟨unsorted-poly-rel ⊆ fully-unsorted-poly-rel⟩
proof -
  have ⟨term-poly-list-rel ×r int-rel ⊆ unsorted-term-poly-list-rel ×r int-rel⟩
  by (auto simp: unsorted-term-poly-list-rel-def term-poly-list-rel-def)
  from list-rel-mono[OF this]
  show ?thesis
  unfolding poly-list-rel-def fully-unsorted-poly-list-rel-def

```

by (auto simp:)
qed

lemma *mult-poly-full-mult-poly-p'*:
assumes $\langle (p, p') \in \text{sorted-poly-rel} \rangle \langle (q, q') \in \text{sorted-poly-rel} \rangle$
shows $\langle \text{mult-poly-full } p \ q \leq \Downarrow (\text{sorted-poly-rel}) (\text{mult-poly-p}' \ p' \ q') \rangle$
unfolding *mult-poly-full-def mult-poly-p'-def*
apply (*refine-rcg full-normalize-poly-normalize-poly-p*
normalize-poly-normalize-poly-p)
apply (*subst RETURN-RES-refine-iff*)
apply (*subst Bex-def*)
apply (*subst mem-Collect-eq*)
apply (*subst conj-commute*)
apply (*rule mult-poly-raw-mult-poly-p[OF assms(1,2)]*)
subgoal
by *blast*
done

definition *add-poly-spec* :: $\langle \rightarrow \rangle$ **where**
 $\langle \text{add-poly-spec } p \ q = \text{SPEC } (\lambda r. \ p + q - r \in \text{ideal polynomial-bool}) \rangle$

definition *add-poly-p'* :: $\langle \rightarrow \rangle$ **where**
 $\langle \text{add-poly-p}' \ p \ q = \text{SPEC}(\lambda r. \ \text{add-poly-p}^{**} (p, q, \{\#\}) (\{\#\}, \{\#\}, r)) \rangle$

lemma *add-poly-l-add-poly-p'*:
assumes $\langle (p, p') \in \text{sorted-poly-rel} \rangle \langle (q, q') \in \text{sorted-poly-rel} \rangle$
shows $\langle \text{add-poly-l } (p, q) \leq \Downarrow (\text{sorted-poly-rel}) (\text{add-poly-p}' \ p' \ q') \rangle$
unfolding *add-poly-p'-def*
apply (*refine-rcg add-poly-l-spec[THEN freq-to-Down-curry-right, THEN order-trans, of - p' q']*)
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal
by *auto*
done

8.7 Correctness

context *poly-embed*
begin

definition *mset-poly-rel* **where**
 $\langle \text{mset-poly-rel} = \{(a, b). \ b = \text{polynomial-of-mset } a\} \rangle$

definition *var-rel* **where**
 $\langle \text{var-rel} = \text{br } \varphi (\lambda \cdot. \ \text{True}) \rangle$

lemma *normalize-poly-p-normalize-poly-spec*:
 $\langle (p, p') \in \text{mset-poly-rel} \implies$
 $\text{SPEC } (\lambda r. \ \text{normalize-poly-p}^{**} \ p \ r) \leq \Downarrow \text{mset-poly-rel } (\text{normalize-poly-spec } \ p') \rangle$
by (*auto simp: mset-poly-rel-def rtranclp-normalize-poly-p-poly-of-mset ideal.span-zero*
normalize-poly-spec-def intro!: RES-refine)

lemma *mult-poly-p'-mult-poly-spec*:
 $\langle (p, p') \in \text{mset-poly-rel} \implies (q, q') \in \text{mset-poly-rel} \implies$
 $\text{mult-poly-p}' \ p \ q \leq \Downarrow \text{mset-poly-rel } (\text{mult-poly-spec } \ p' \ q') \rangle$

```

unfolding mult-poly-p'-def mult-poly-spec-def
apply refine-rcg
apply (auto simp: mset-poly-rel-def dest!: rtranclp-mult-poly-p-mult-ideal-final)
apply (intro RES-refine)
using ideal.span-add-eq2 ideal.span-zero
by (fastforce dest!: rtranclp-normalize-poly-p-poly-of-mset intro: ideal.span-add-eq2)

```

```

lemma add-poly-p'-add-poly-spec:
   $\langle (p, p') \in \text{mset-poly-rel} \implies (q, q') \in \text{mset-poly-rel} \implies$ 
   $\text{add-poly-p}' p q \leq \Downarrow \text{mset-poly-rel} (\text{add-poly-spec } p' q') \rangle$ 
unfolding add-poly-p'-def add-poly-spec-def
apply (auto simp: mset-poly-rel-def dest!: rtranclp-add-poly-p-polynomial-of-mset-full)
apply (intro RES-refine)
apply (auto simp: rtranclp-add-poly-p-polynomial-of-mset-full ideal.span-zero)
done

```

end

```

definition weak-equality-l ::  $\langle \text{llist-polynomial} \Rightarrow \text{llist-polynomial} \Rightarrow \text{bool nres} \rangle$  where
   $\langle \text{weak-equality-l } p q = \text{RETURN } (p = q) \rangle$ 

```

```

definition weak-equality ::  $\langle \text{int mpoly} \Rightarrow \text{int mpoly} \Rightarrow \text{bool nres} \rangle$  where
   $\langle \text{weak-equality } p q = \text{SPEC } (\lambda r. r \longrightarrow p = q) \rangle$ 

```

```

definition weak-equality-spec ::  $\langle \text{mset-polynomial} \Rightarrow \text{mset-polynomial} \Rightarrow \text{bool nres} \rangle$  where
   $\langle \text{weak-equality-spec } p q = \text{SPEC } (\lambda r. r \longrightarrow p = q) \rangle$ 

```

```

lemma term-poly-list-rel-same-rightD:
   $\langle (a, aa) \in \text{term-poly-list-rel} \implies (a, ab) \in \text{term-poly-list-rel} \implies aa = ab \rangle$ 
  by (auto simp: term-poly-list-rel-def)

```

```

lemma list-rel-term-poly-list-rel-same-rightD:
   $\langle (xa, y) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$ 
   $(xa, ya) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$ 
   $y = ya \rangle$ 
by (induction xa arbitrary: y ya)
  (auto simp: list-rel-split-right-iff
  dest: term-poly-list-rel-same-rightD)

```

```

lemma weak-equality-l-weak-equality-spec:
   $\langle (\text{uncurry } \text{weak-equality-l}, \text{uncurry } \text{weak-equality-spec}) \in$ 
   $\text{sorted-poly-rel} \times_r \text{sorted-poly-rel} \rightarrow_f \langle \text{bool-rel} \rangle \text{nres-rel} \rangle$ 
by (intro frefI nres-relI)
  (auto simp: weak-equality-l-def weak-equality-spec-def
  sorted-poly-list-rel-wrt-def list-mset-rel-def br-def
  dest: list-rel-term-poly-list-rel-same-rightD)

```

end

theory *PAC-Misc*

imports *Main*

begin

I believe this should be added to the simplifier by default...

lemma *Collect-eq-comp'*:
 $\{(x, y). P x y\} \circ \{(c, a). c = f a\} = \{(x, a). P x (f a)\}$
by *auto*

lemma *in-set-conv-iff*:
 $x \in \text{set } (\text{take } n \text{ } xs) \longleftrightarrow (\exists i < n. i < \text{length } xs \wedge xs ! i = x)$
by (*metis in-set-conv-nth length-take min-less-iff-conj nth-take*)

lemma *in-set-take-conv-nth*:
 $x \in \text{set } (\text{take } n \text{ } xs) \longleftrightarrow (\exists i < \min n (\text{length } xs). xs ! i = x)$
by (*simp add: in-set-conv-iff*)

lemma *in-set-remove1D*:
 $a \in \text{set } (\text{remove1 } x \text{ } xs) \implies a \in \text{set } xs$
by (*meson notin-set-remove1*)

end

theory *PAC-Checker*
imports *PAC-Polynomials-Operations*
PAC-Checker-Specification
PAC-Map-Rel
Show.Show
Show.Show-Instances
PAC-Misc

begin

9 Executable Checker

In this layer we finally refine the checker to executable code.

9.1 Definitions

Compared to the previous layer, we add an error message when an error is discovered. We do not attempt to prove anything on the error message (neither that there really is an error, nor that the error message is correct).

Extended error message datatype *'a code-status =*
is-cfailed: CFAILED (the-error: 'a) |
CSUCCESS |
is-cfound: CFOUND

In the following function, we merge errors. We will never merge an error message with another error message; hence we do not attempt to concatenate error messages.

fun *merge-cstatus where*
 $\langle \text{merge-cstatus } (CFAILED a) - = CFAILED a \rangle |$
 $\langle \text{merge-cstatus } - (CFAILED a) = CFAILED a \rangle |$
 $\langle \text{merge-cstatus } CFOUND - = CFOUND \rangle |$
 $\langle \text{merge-cstatus } - CFOUND = CFOUND \rangle |$
 $\langle \text{merge-cstatus } - - = CSUCCESS \rangle$

definition *code-status-status-rel* :: $\langle ('a \text{ code-status} \times \text{status}) \text{ set} \rangle$ **where**
 $\langle \text{code-status-status-rel} =$

$\{(CFOUND, FOUND), (CSUCCESS, SUCCESS)\} \cup$
 $\{(CFAILED\ a, FAILED)\mid a.\ True\}$

lemma *in-code-status-status-rel-iff[simp]*:

$\langle (CFOUND, b) \in \text{code-status-status-rel} \longleftrightarrow b = FOUND \rangle$
 $\langle (a, FOUND) \in \text{code-status-status-rel} \longleftrightarrow a = CFOUND \rangle$
 $\langle (CSUCCESS, b) \in \text{code-status-status-rel} \longleftrightarrow b = SUCCESS \rangle$
 $\langle (a, SUCCESS) \in \text{code-status-status-rel} \longleftrightarrow a = CSUCCESS \rangle$
 $\langle (a, FAILED) \in \text{code-status-status-rel} \longleftrightarrow \text{is-cfailed } a \rangle$
 $\langle (CFAILED\ C, b) \in \text{code-status-status-rel} \longleftrightarrow b = FAILED \rangle$
by (*cases a; cases b; auto simp: code-status-status-rel-def; fail*) $+$

Refinement relation **fun** *pac-step-rel-raw* :: $\langle ('olbl \times 'lbl)\ \text{set} \Rightarrow ('a \times 'b)\ \text{set} \Rightarrow ('c \times 'd)\ \text{set} \Rightarrow$
 $('a, 'c, 'olbl)\ \text{pac-step} \Rightarrow ('b, 'd, 'lbl)\ \text{pac-step} \Rightarrow \text{bool} \rangle$ **where**

$\langle \text{pac-step-rel-raw } R1\ R2\ R3\ (\text{Add } p1\ p2\ i\ r)\ (\text{Add } p1'\ p2'\ i'\ r') \longleftrightarrow$
 $(p1, p1') \in R1 \wedge (p2, p2') \in R2 \wedge (i, i') \in R3 \wedge$
 $(r, r') \in R2 \rangle$ |
 $\langle \text{pac-step-rel-raw } R1\ R2\ R3\ (\text{Mult } p1\ p2\ i\ r)\ (\text{Mult } p1'\ p2'\ i'\ r') \longleftrightarrow$
 $(p1, p1') \in R1 \wedge (p2, p2') \in R2 \wedge (i, i') \in R3 \wedge$
 $(r, r') \in R2 \rangle$ |
 $\langle \text{pac-step-rel-raw } R1\ R2\ R3\ (\text{Del } p1)\ (\text{Del } p1') \longleftrightarrow$
 $(p1, p1') \in R1 \rangle$ |
 $\langle \text{pac-step-rel-raw } R1\ R2\ R3\ (\text{Extension } i\ x\ p1)\ (\text{Extension } j\ x'\ p1') \longleftrightarrow$
 $(i, j) \in R1 \wedge (x, x') \in R3 \wedge (p1, p1') \in R2 \rangle$ |
 $\langle \text{pac-step-rel-raw } R1\ R2\ R3\ -\ - \longleftrightarrow \text{False} \rangle$

fun *pac-step-rel-assn* :: $\langle ('olbl \Rightarrow 'lbl \Rightarrow \text{assn}) \Rightarrow ('a \Rightarrow 'b \Rightarrow \text{assn}) \Rightarrow ('c \Rightarrow 'd \Rightarrow \text{assn}) \Rightarrow ('a, 'c,$
 $'olbl)\ \text{pac-step} \Rightarrow ('b, 'd, 'lbl)\ \text{pac-step} \Rightarrow \text{assn} \rangle$ **where**

$\langle \text{pac-step-rel-assn } R1\ R2\ R3\ (\text{Add } p1\ p2\ i\ r)\ (\text{Add } p1'\ p2'\ i'\ r') =$
 $R1\ p1\ p1' * R1\ p2\ p2' * R1\ i\ i' * R2\ r\ r' \rangle$ |
 $\langle \text{pac-step-rel-assn } R1\ R2\ R3\ (\text{Mult } p1\ p2\ i\ r)\ (\text{Mult } p1'\ p2'\ i'\ r') =$
 $R1\ p1\ p1' * R2\ p2\ p2' * R1\ i\ i' * R2\ r\ r' \rangle$ |
 $\langle \text{pac-step-rel-assn } R1\ R2\ R3\ (\text{Del } p1)\ (\text{Del } p1') =$
 $R1\ p1\ p1' \rangle$ |
 $\langle \text{pac-step-rel-assn } R1\ R2\ R3\ (\text{Extension } i\ x\ p1)\ (\text{Extension } i'\ x'\ p1') =$
 $R1\ i\ i' * R3\ x\ x' * R2\ p1\ p1' \rangle$ |
 $\langle \text{pac-step-rel-assn } R1\ R2\ -\ -\ - = \text{false} \rangle$

lemma *pac-step-rel-assn-alt-def*:

$\langle \text{pac-step-rel-assn } R1\ R2\ R3\ x\ y =$
case (x, y) *of*
 $(\text{Add } p1\ p2\ i\ r, \text{Add } p1'\ p2'\ i'\ r') \Rightarrow$
 $R1\ p1\ p1' * R1\ p2\ p2' * R1\ i\ i' * R2\ r\ r'$
| $(\text{Mult } p1\ p2\ i\ r, \text{Mult } p1'\ p2'\ i'\ r') \Rightarrow$
 $R1\ p1\ p1' * R2\ p2\ p2' * R1\ i\ i' * R2\ r\ r'$
| $(\text{Del } p1, \text{Del } p1') \Rightarrow R1\ p1\ p1'$
| $(\text{Extension } i\ x\ p1, \text{Extension } i'\ x'\ p1') \Rightarrow R1\ i\ i' * R3\ x\ x' * R2\ p1\ p1'$
| $- \Rightarrow \text{false} \rangle$
by (*auto split: pac-step.splits*)

Addition checking **definition** *error-msg* **where**

$\langle \text{error-msg } i\ msg = \text{CFAILED } ("s\ \text{CHECKING failed at line } " @ \text{show } i @ " \text{ with error } " @ msg) \rangle$

definition *error-msg-notin-dom-err* **where**
 ⟨*error-msg-notin-dom-err* = "notin domain"⟩

definition *error-msg-notin-dom* :: ⟨nat ⇒ string⟩ **where**
 ⟨*error-msg-notin-dom* i = show i @ *error-msg-notin-dom-err*⟩

definition *error-msg-reused-dom* **where**
 ⟨*error-msg-reused-dom* i = show i @ "already in domain"⟩

definition *error-msg-not-equal-dom* **where**
 ⟨*error-msg-not-equal-dom* p q pq r = show p @ " + " @ show q @ " = " @ show pq @ " not equal"
 @ show r⟩

definition *check-not-equal-dom-err* :: ⟨l_{ist}-polynomial ⇒ l_{ist}-polynomial ⇒ l_{ist}-polynomial ⇒ l_{ist}-polynomial
 ⇒ string nres⟩ **where**
 ⟨*check-not-equal-dom-err* p q pq r = SPEC (λ-. True)⟩

definition *vars-l_{ist}* :: ⟨l_{ist}-polynomial ⇒ string set⟩ **where**
 ⟨*vars-l_{ist}* xs = ⋃(set 'fst 'set xs)⟩

definition *check-addition-l* :: ⟨- ⇒ - ⇒ string set ⇒ nat ⇒ nat ⇒ nat ⇒ l_{ist}-polynomial ⇒ string
 code-status nres⟩ **where**
 ⟨*check-addition-l* spec A V p q i r = do {
 let b = p ∈# dom-m A ∧ q ∈# dom-m A ∧ i ∉# dom-m A ∧ vars-l_{ist} r ⊆ V;
 if ¬b
 then RETURN (error-msg i ((if p ∉# dom-m A then error-msg-notin-dom p else []) @ (if q ∉#
 dom-m A then error-msg-notin-dom p else [])) @
 (if i ∈# dom-m A then error-msg-reused-dom p else []))
 else do {
 ASSERT (p ∈# dom-m A);
 let p = the (fmlookup A p);
 ASSERT (q ∈# dom-m A);
 let q = the (fmlookup A q);
 pq ← add-poly-l (p, q);
 b ← weak-equality-l pq r;
 b' ← weak-equality-l r spec;
 if b then (if b' then RETURN CFOUND else RETURN CSUCCESS)
 else do {
 c ← check-not-equal-dom-err p q pq r;
 RETURN (error-msg i c)}
 }
 }
 }⟩

Multiplication checking **definition** *check-mult-l-dom-err* :: ⟨bool ⇒ nat ⇒ bool ⇒ nat ⇒ string
 nres⟩ **where**
 ⟨*check-mult-l-dom-err* p-notin p i-already i = SPEC (λ-. True)⟩

definition *check-mult-l-mult-err* :: ⟨l_{ist}-polynomial ⇒ l_{ist}-polynomial ⇒ l_{ist}-polynomial ⇒ l_{ist}-polynomial
 ⇒ string nres⟩ **where**
 ⟨*check-mult-l-mult-err* p q pq r = SPEC (λ-. True)⟩

definition *check-mult-l* :: $\langle - \Rightarrow - \Rightarrow - \Rightarrow \text{nat} \Rightarrow \text{l-list-polynomial} \Rightarrow \text{nat} \Rightarrow \text{l-list-polynomial} \Rightarrow \text{string code-status nres} \rangle$ **where**
 $\langle \text{check-mult-l spec } A \mathcal{V} p q i r = \text{do} \{$
 $\text{let } b = p \in\# \text{dom-m } A \wedge i \notin\# \text{dom-m } A \wedge \text{vars-l-list } q \subseteq \mathcal{V} \wedge \text{vars-l-list } r \subseteq \mathcal{V};$
 $\text{if } \neg b$
 $\text{then do} \{$
 $c \leftarrow \text{check-mult-l-dom-err } (p \notin\# \text{dom-m } A) p (i \in\# \text{dom-m } A) i;$
 $\text{RETURN } (\text{error-msg } i c)$
 $\text{else do} \{$
 $\text{ASSERT } (p \in\# \text{dom-m } A);$
 $\text{let } p = \text{the } (\text{fmlookup } A p);$
 $pq \leftarrow \text{mult-poly-full } p q;$
 $b \leftarrow \text{weak-equality-l } pq r;$
 $b' \leftarrow \text{weak-equality-l } r \text{ spec};$
 $\text{if } b \text{ then } (\text{if } b' \text{ then } \text{RETURN } \text{CFOUND} \text{ else } \text{RETURN } \text{CSUCCESS}) \text{ else do} \{$
 $c \leftarrow \text{check-mult-l-mult-err } p q pq r;$
 $\text{RETURN } (\text{error-msg } i c)$
 $\}$
 $\}$
 $\}$
 \rangle

Deletion checking definition *check-del-l* :: $\langle - \Rightarrow - \Rightarrow \text{nat} \Rightarrow \text{string code-status nres} \rangle$ **where**
 $\langle \text{check-del-l spec } A p = \text{RETURN } \text{CSUCCESS} \rangle$

Extension checking definition *check-extension-l-dom-err* :: $\langle \text{nat} \Rightarrow \text{string nres} \rangle$ **where**
 $\langle \text{check-extension-l-dom-err } p = \text{SPEC } (\lambda-. \text{True}) \rangle$

definition *check-extension-l-no-new-var-err* :: $\langle \text{l-list-polynomial} \Rightarrow \text{string nres} \rangle$ **where**
 $\langle \text{check-extension-l-no-new-var-err } p = \text{SPEC } (\lambda-. \text{True}) \rangle$

definition *check-extension-l-new-var-multiple-err* :: $\langle \text{string} \Rightarrow \text{l-list-polynomial} \Rightarrow \text{string nres} \rangle$ **where**
 $\langle \text{check-extension-l-new-var-multiple-err } v p = \text{SPEC } (\lambda-. \text{True}) \rangle$

definition *check-extension-l-side-cond-err*
:: $\langle \text{string} \Rightarrow \text{l-list-polynomial} \Rightarrow \text{l-list-polynomial} \Rightarrow \text{l-list-polynomial} \Rightarrow \text{string nres} \rangle$
where
 $\langle \text{check-extension-l-side-cond-err } v p p' q = \text{SPEC } (\lambda-. \text{True}) \rangle$

definition *check-extension-l*
:: $\langle - \Rightarrow - \Rightarrow \text{string set} \Rightarrow \text{nat} \Rightarrow \text{string} \Rightarrow \text{l-list-polynomial} \Rightarrow (\text{string code-status}) \text{ nres} \rangle$
where
 $\langle \text{check-extension-l spec } A \mathcal{V} i v p = \text{do} \{$
 $\text{let } b = i \notin\# \text{dom-m } A \wedge v \notin \mathcal{V} \wedge ([v], -1) \in \text{set } p;$
 $\text{if } \neg b$
 $\text{then do} \{$
 $c \leftarrow \text{check-extension-l-dom-err } i;$
 $\text{RETURN } (\text{error-msg } i c)$
 $\}$ $\text{else do} \{$
 $\text{let } p' = \text{remove1 } ([v], -1) p;$
 $\text{let } b = \text{vars-l-list } p' \subseteq \mathcal{V};$
 $\text{if } \neg b$
 $\text{then do} \{$


```

show ?thesis
  unfolding p
  apply (subst coeff-minus[symmetric], subst not-in-vars-coeff0)
  using that unfolding q-def[symmetric]
  by (auto simp flip: coeff-minus simp: not-in-vars-coeff0
      Var.abs-eq Var0-def simp flip: monom.abs-eq
      simp: not-in-vars-coeff0 MPoly-Type.coeff-def
      Var.abs-eq Var0-def lookup-single-eq monom.rep-eq)
qed
have [simp]: ⟨vars (p - Var ab) = vars (Var ab - p)⟩ for ab
  using vars-uminus[of ⟨p - Var ab⟩]
  by simp
show ?thesis
  unfolding check-extension-def
  apply (auto 5 5 simp: check-extension-def weak-equality-def
      mult-poly-spec-def field-simps
      add-poly-spec-def power2-eq-square cong: if-cong
      intro!: intro-spec-refine[where R=Id, simplified]
      split: option.splits dest: ideal.span-add)
  done
qed

```

lemma RES-RES-RETURN-RES: $\langle RES\ A \gg (\lambda T. RES\ (f\ T)) = RES\ (\bigcup (f\ 'A)) \rangle$
by (auto simp: pw-eq-iff refine-pw-simps)

lemma check-add-alt-def:

```

⟨check-add A  $\mathcal{V}$  p q i r  $\geq$ 
  do {
    b  $\leftarrow$  SPEC( $\lambda b. b \longrightarrow p \in \# \text{dom-}m\ A \wedge q \in \# \text{dom-}m\ A \wedge i \notin \# \text{dom-}m\ A \wedge \text{vars } r \subseteq \mathcal{V}$ );
    if  $\neg b$ 
    then RETURN False
    else do {
      ASSERT (p  $\in \# \text{dom-}m\ A$ );
      let p = the (fmlookup A p);
      ASSERT (q  $\in \# \text{dom-}m\ A$ );
      let q = the (fmlookup A q);
      pq  $\leftarrow$  add-poly-spec p q;
      eq  $\leftarrow$  weak-equality pq r;
      RETURN eq
    }
  }
  ⟩ (is  $\langle - \geq ?A \rangle$ )

```

proof –

```

have check-add-alt-def: ⟨check-add A  $\mathcal{V}$  p q i r = do {
  b  $\leftarrow$  SPEC( $\lambda b. b \longrightarrow p \in \# \text{dom-}m\ A \wedge q \in \# \text{dom-}m\ A \wedge i \notin \# \text{dom-}m\ A \wedge \text{vars } r \subseteq \mathcal{V}$ );
  if  $\neg b$  then SPEC( $\lambda b. b \longrightarrow p \in \# \text{dom-}m\ A \wedge q \in \# \text{dom-}m\ A \wedge i \notin \# \text{dom-}m\ A \wedge \text{vars } r \subseteq \mathcal{V} \wedge$ 
    the (fmlookup A p) + the (fmlookup A q) - r  $\in$  ideal polynomial-bool)
  else
    SPEC( $\lambda b. b \longrightarrow p \in \# \text{dom-}m\ A \wedge q \in \# \text{dom-}m\ A \wedge i \notin \# \text{dom-}m\ A \wedge \text{vars } r \subseteq \mathcal{V} \wedge$ 
      the (fmlookup A p) + the (fmlookup A q) - r  $\in$  ideal polynomial-bool)}
  (is  $\langle - = ?B \rangle$ )
by (auto simp: check-add-def RES-RES-RETURN-RES)
have  $\langle ?A \leq \Downarrow Id\ (\text{check-add } A\ \mathcal{V}\ p\ q\ i\ r) \rangle$ 
apply refine-vcg

```

```

apply ((auto simp: check-add-alt-def weak-equality-def
  add-poly-spec-def RES-RES-RETURN-RES summarize-ASSERT-conv
  cong: if-cong
  intro!: ideal.span-zero;fail)+)
done
then show ?thesis
  unfolding check-add-alt-def[symmetric]
  by simp
qed

```

```

lemma check-mult-alt-def:
  ⟨check-mult A V p q i r ≥
  do {
    b ← SPEC(λb. b → p ∈# dom-m A ∧ i ∉# dom-m A ∧ vars q ⊆ V ∧ vars r ⊆ V);
    if ¬b
    then RETURN False
    else do {
      ASSERT (p ∈# dom-m A);
      let p = the (fmlookup A p);
      pq ← mult-poly-spec p q;
      p ← weak-equality pq r;
      RETURN p
    }
  }⟩
unfolding check-mult-def
apply (rule refine-IdD)
by refine-vcg
  (auto simp: check-mult-def weak-equality-def
  mult-poly-spec-def RES-RES-RETURN-RES
  intro!: ideal.span-zero
  exI[of - (the (fmlookup A p) * q)])

```

```

primrec insort-key-rel :: ('b ⇒ 'b ⇒ bool) ⇒ 'b ⇒ 'b list ⇒ 'b list where
insort-key-rel f x [] = [x] |
insort-key-rel f x (y#ys) =
  (if f x y then (x#y#ys) else y#(insort-key-rel f x ys))

```

```

lemma set-insort-key-rel[simp]: ⟨set (insort-key-rel R x xs) = insert x (set xs)⟩
by (induction xs)
  auto

```

```

lemma sorted-wrt-insort-key-rel:
  ⟨total-on R (insert x (set xs)) ⇒ transp R ⇒ reflp R ⇒
  sorted-wrt R xs ⇒ sorted-wrt R (insort-key-rel R x xs)⟩
by (induction xs)
  (auto dest: transpD reflpD simp: Restricted-Predicates.total-on-def)

```

```

lemma sorted-wrt-insort-key-rel2:
  ⟨total-on R (insert x (set xs)) ⇒ transp R ⇒ x ∉ set xs ⇒
  sorted-wrt R xs ⇒ sorted-wrt R (insort-key-rel R x xs)⟩
by (induction xs)
  (auto dest: transpD simp: Restricted-Predicates.total-on-def in-mono)

```

Step checking **definition** *PAC-checker-l-step* :: ⟨*- ⇒ string code-status × string set × - ⇒ (l-list-polynomial,*
string, nat) pac-step ⇒ -⟩ **where**

```

⟨PAC-checker-l-step = (λspec (st', V, A) st. case st of
  Add - - - ⇒
    do {
      r ← full-normalize-poly (pac-res st);
      eq ← check-addition-l spec A V (pac-src1 st) (pac-src2 st) (new-id st) r;
      let - = eq;
      if ¬is-cfailed eq
      then RETURN (merge-cstatus st' eq,
        V, fmupd (new-id st) r A)
      else RETURN (eq, V, A)
    }
  | Del - ⇒
    do {
      eq ← check-del-l spec A (pac-src1 st);
      let - = eq;
      if ¬is-cfailed eq
      then RETURN (merge-cstatus st' eq, V, fmdrop (pac-src1 st) A)
      else RETURN (eq, V, A)
    }
  | Mult - - - ⇒
    do {
      r ← full-normalize-poly (pac-res st);
      q ← full-normalize-poly (pac-mult st);
      eq ← check-mult-l spec A V (pac-src1 st) q (new-id st) r;
      let - = eq;
      if ¬is-cfailed eq
      then RETURN (merge-cstatus st' eq,
        V, fmupd (new-id st) r A)
      else RETURN (eq, V, A)
    }
  | Extension - - - ⇒
    do {
      r ← full-normalize-poly (([new-var st], -1) # (pac-res st));
      (eq) ← check-extension-l spec A V (new-id st) (new-var st) r;
      if ¬is-cfailed eq
      then do {
        RETURN (st',
          insert (new-var st) V, fmupd (new-id st) r A)}
      else RETURN (eq, V, A)
    }
),

```

lemma *pac-step-rel-raw-def*:

⟨⟨K, V, R⟩ pac-step-rel-raw = pac-step-rel-raw K V R⟩

by (auto intro!: ext simp: relAPP-def)

definition *mononoms-equal-up-to-reorder* **where**

⟨mononoms-equal-up-to-reorder xs ys ↔

map (λ(a, b). (mset a, b)) xs = map (λ(a, b). (mset a, b)) ys⟩

definition *normalize-poly-l* **where**

⟨normalize-poly-l p = SPEC (λp'.

normalize-poly-p* ((λ(a, b). (mset a, b)) '# mset p) ((λ(a, b). (mset a, b)) '# mset p') ∧

0 ∉# snd '# mset p' ∧

$sorted-wrt (rel2p (term-order-rel \times_r int-rel)) p' \wedge$
 $(\forall x \in monoms p'. sorted-wrt (rel2p var-order-rel) x))\rangle$

definition *remap-polys-l-dom-err* :: $\langle string nres \rangle$ **where**
 $\langle remap-polys-l-dom-err = SPEC (\lambda-. True) \rangle$

definition *remap-polys-l* :: $\langle llist-polynomial \Rightarrow string set \Rightarrow (nat, llist-polynomial) fmap \Rightarrow$
 $(- code-status \times string set \times (nat, llist-polynomial) fmap) nres \rangle$ **where**
 $\langle remap-polys-l spec = (\lambda V A. do\{$
 $dom \leftarrow SPEC(\lambda dom. set-mset (dom-m A) \subseteq dom \wedge finite dom);$
 $failed \leftarrow SPEC(\lambda-::bool. True);$
 $if failed$
 $then do \{$
 $c \leftarrow remap-polys-l-dom-err;$
 $RETURN (error-msg (0 :: nat) c, V, fmempty)$
 $\}$
 $else do \{$
 $(b, V, A) \leftarrow FOREACH dom$
 $(\lambda i (b, V, A').$
 $if $i \in \# dom-m A$$
 $then do \{$
 $p \leftarrow full-normalize-poly (the (fmlookup A i));$
 $eq \leftarrow weak-equality-l p spec;$
 $V \leftarrow RETURN(V \cup vars-llist (the (fmlookup A i)));$
 $RETURN(b \vee eq, V, fmupd i p A')$
 $\} else RETURN (b, V, A')$
 $(False, V, fmempty);$
 $RETURN (if b then CFOUND else CSUCCESS, V, A)$
 $\}\}\rangle$

definition *PAC-checker-l* **where**
 $\langle PAC-checker-l spec A b st = do \{$
 $(S, -) \leftarrow WHILE_T$
 $(\lambda((b, A), n). \neg is-cfailed b \wedge n \neq [])$
 $(\lambda((bA), n). do \{$
 $ASSERT($n \neq []$);$
 $S \leftarrow PAC-checker-l-step spec bA (hd n);$
 $RETURN (S, tl n)$
 $\})$
 $((b, A), st);$
 $RETURN S$
 $\}\rangle$

9.2 Correctness

We now enter the locale to reason about polynomials directly.

context *poly-embed*
begin

abbreviation *pac-step-rel* **where**
 $\langle pac-step-rel \equiv p2rel (\langle Id, fully-unsorted-poly-rel \ O mset-poly-rel, var-rel \rangle pac-step-rel-raw) \rangle$

abbreviation *fmap-polys-rel* **where**

$\langle \text{fmap-polys-rel} \equiv \langle \text{nat-rel}, \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle \text{fmap-rel} \rangle$

lemma

$\langle \text{normalize-poly-p } s0 \ s \implies$
 $(s0, p) \in \text{mset-poly-rel} \implies$
 $(s, p) \in \text{mset-poly-rel} \rangle$
by (*auto simp: mset-poly-rel-def normalize-poly-p-poly-of-mset*)

lemma vars-poly-of-vars:

$\langle \text{vars } (\text{poly-of-vars } a :: \text{int mpoly}) \subseteq (\varphi \text{ ' set-mset } a) \rangle$
by (*induction a*)
(auto simp: vars-mult-Var)

lemma vars-polynomial-of-mset:

$\langle \text{vars } (\text{polynomial-of-mset } za) \subseteq \bigcup (\text{image } \varphi \text{ ' (set-mset } o \text{ fst) ' set-mset } za) \rangle$
apply (*induction za*)
using *vars-poly-of-vars*
by (*fastforce elim!: in-vars-addE simp: vars-mult-Const split: if-splits*) $+$

lemma fully-unsorted-poly-rel-vars-subset-vars-llist:

$\langle (A, B) \in \text{fully-unsorted-poly-rel } O \text{ mset-poly-rel} \implies \text{vars } B \subseteq \varphi \text{ ' vars-llist } A \rangle$
by (*auto simp: fully-unsorted-poly-list-rel-def mset-poly-rel-def*
set-rel-def var-rel-def br-def vars-llist-def list-rel-append2 list-rel-append1
list-rel-split-right-iff list-mset-rel-def image-iff
unsorted-term-poly-list-rel-def list-rel-split-left-iff
dest!: set-rev-mp[OF - vars-polynomial-of-mset] split-list
dest: multi-member-split
dest: arg-cong[of <mset -> <add-mset - -> set-mset])

lemma fully-unsorted-poly-rel-extend-vars:

$\langle (A, B) \in \text{fully-unsorted-poly-rel } O \text{ mset-poly-rel} \implies$
 $(x1c, x1a) \in \langle \text{var-rel} \rangle \text{set-rel} \implies$
 $\text{RETURN } (x1c \cup \text{vars-llist } A)$
 $\leq \Downarrow ((\text{var-rel}) \text{set-rel})$
 $(\text{SPEC } ((\subseteq) (x1a \cup \text{vars } (B)))) \rangle$
using *fully-unsorted-poly-rel-vars-subset-vars-llist*[of $A \ B$]
apply (*subst RETURN-RES-refine-iff*)
apply *clarsimp*
apply (*rule exI*[of - $\langle x1a \cup \varphi \text{ ' vars-llist } A \rangle$])
apply (*auto simp: set-rel-def var-rel-def br-def*
dest: fully-unsorted-poly-rel-vars-subset-vars-llist)
done

lemma remap-polys-l-remap-polys:

assumes
 $AB: \langle (A, B) \in \langle \text{nat-rel}, \text{fully-unsorted-poly-rel } O \text{ mset-poly-rel} \rangle \text{fmap-rel} \rangle$ **and**
 $\text{spec}: \langle (\text{spec}, \text{spec}') \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle$ **and**
 $V: \langle (\mathcal{V}, \mathcal{V}') \in \langle \text{var-rel} \rangle \text{set-rel} \rangle$
shows $\langle \text{remap-polys-l spec } \mathcal{V} \ A \leq$
 $\Downarrow (\text{code-status-status-rel } \times_r \langle \text{var-rel} \rangle \text{set-rel } \times_r \text{fmap-polys-rel}) (\text{remap-polys spec}' \ \mathcal{V}' \ B) \rangle$
(is $\langle - \leq \Downarrow ?R - \rangle$)

proof –

have $1: \langle \text{inj-on id } (\text{dom} :: \text{nat set}) \rangle$ **for** dom
by *auto*
have $H: \langle x \in \# \text{dom-m } A \implies$

$(\bigwedge p. (the (fmlookup A x), p) \in fully\text{-unsorted-poly-rel} \implies$
 $(p, the (fmlookup B x)) \in mset\text{-poly-rel} \implies thesis) \implies$
 $thesis \rangle$ **for** x *thesis*
using *fmap-rel-nat-the-fmlookup*[*OF AB*, *of x x*] *fmap-rel-nat-rel-dom-m*[*OF AB*] **by** *auto*
have *full-normalize-poly*: $\langle full\text{-normalize-poly} (the (fmlookup A x))$
 $\leq \Downarrow (sorted\text{-poly-rel } O \ mset\text{-poly-rel})$
 $(SPEC$
 $(\lambda p. the (fmlookup B x') - p \in More\text{-Modules.ideal polynomial-bool} \wedge$
 $vars \ p \subseteq vars (the (fmlookup B x')))) \rangle$
if $x\text{-dom}$: $\langle x \in \# \ dom\text{-}m \ A \rangle$ **and** $\langle (x, x') \in Id \rangle$ **for** $x \ x'$
apply (*rule* $H[OF \ x\text{-dom}]$)
subgoal **for** p
apply (*rule* *full-normalize-poly-normalize-poly-p*[*THEN order-trans*])
apply *assumption*
subgoal
using *that*(\emptyset) **apply** –
unfolding *conc-fun-chain*[*symmetric*]
by (*rule* *ref-two-step'*, *rule* *RES-refine*)
 $(auto \ simp: \ rtranclp\text{-normalize-poly-p-poly-of-mset}$
 $mset\text{-poly-rel-def ideal.span-zero})$
done
done

have H' : $\langle (p, pa) \in sorted\text{-poly-rel } O \ mset\text{-poly-rel} \implies$
 $weak\text{-equality-l } p \ spec \leq SPEC (\lambda eqa. eqa \longrightarrow pa = spec') \rangle$ **for** $p \ pa$
using *spec* **by** (*auto simp: weak-equality-l-def weak-equality-spec-def*
 $list\text{-mset-rel-def br-def mset-poly-rel-def}$
 $dest: list\text{-rel-term-poly-list-rel-same-rightD sorted-poly-list-relD}$)

have *emp*: $\langle (\mathcal{V}, \mathcal{V}') \in (var\text{-rel})set\text{-rel} \implies$
 $((False, \mathcal{V}, fmempty), False, \mathcal{V}', fmempty) \in bool\text{-rel} \times_r (var\text{-rel})set\text{-rel} \times_r fmap\text{-polys-rel} \rangle$ **for** $\mathcal{V} \ \mathcal{V}'$
by *auto*
show *?thesis*
using *assms*
unfolding *remap-polys-l-def remap-polys-l-dom-err-def*
 $remap\text{-polys-def prod.case}$
apply (*refine-rcg full-normalize-poly fmap-rel-fmupd-fmap-rel*)
subgoal
by *auto*
subgoal
by *auto*
subgoal
by (*auto simp: error-msg-def*)
apply (*rule* 1)
subgoal **by** *auto*
apply (*rule* *emp*)
subgoal
using V **by** *auto*
subgoal **by** *auto*
subgoal **by** *auto*
subgoal **by** (*rule* H')
apply (*rule* *fully-unsorted-poly-rel-extend-vars*)
subgoal **by** (*auto intro!: fmap-rel-nat-the-fmlookup*)
subgoal **by** (*auto intro!: fmap-rel-fmupd-fmap-rel*)
subgoal **by** (*auto intro!: fmap-rel-fmupd-fmap-rel*)

subgoal by auto
 subgoal by auto
 done
 qed

lemma *fref-to-Down-curry*:

$\langle \text{uncurry } f, \text{uncurry } g \rangle \in [P]_f A \rightarrow \langle B \rangle \text{nres-rel} \implies$
 $\langle \bigwedge x x' y y'. P(x', y') \implies ((x, y), (x', y')) \in A \implies f x y \leq \Downarrow B (g x' y') \rangle$
unfolding *fref-def uncurry-def nres-rel-def*
by *auto*

lemma *weak-equality-spec-weak-equality*:

$\langle (p, p') \in \text{mset-poly-rel} \implies$
 $(r, r') \in \text{mset-poly-rel} \implies$
 $\text{weak-equality-spec } p \ r \leq \text{weak-equality } p' \ r' \rangle$
unfolding *weak-equality-spec-def weak-equality-def*
by (*auto simp: mset-poly-rel-def*)

lemma *weak-equality-l-weak-equality-l'[refine]*:

$\langle \text{weak-equality-l } p \ q \leq \Downarrow \text{bool-rel } (\text{weak-equality } p' \ q') \rangle$
if $\langle (p, p') \in \text{sorted-poly-rel } O \ \text{mset-poly-rel} \rangle$
 $\langle (q, q') \in \text{sorted-poly-rel } O \ \text{mset-poly-rel} \rangle$
for $p \ p' \ q \ q'$
using *that*
by (*auto intro!*: *weak-equality-l-weak-equality-spec*[*THEN fref-to-Down-curry, THEN order-trans*]
ref-two-step'
weak-equality-spec-weak-equality
simp flip: conc-fun-chain)

lemma *error-msg-ne-SUCCESS*[*iff*]:

$\langle \text{error-msg } i \ m \neq \text{CSUCCESS} \rangle$
 $\langle \text{error-msg } i \ m \neq \text{CFOUND} \rangle$
 $\langle \text{is-cfailed } (\text{error-msg } i \ m) \rangle$
 $\langle \neg \text{is-cfound } (\text{error-msg } i \ m) \rangle$
by (*auto simp: error-msg-def*)

lemma *sorted-poly-rel-vars-llist*:

$\langle (r, r') \in \text{sorted-poly-rel } O \ \text{mset-poly-rel} \implies$
 $\text{vars } r' \subseteq \varphi \ \langle \text{vars-llist } r \rangle$
apply (*auto simp: mset-poly-rel-def*
set-rel-def var-rel-def br-def vars-llist-def list-rel-append2 list-rel-append1
list-rel-split-right-iff list-mset-rel-def image-iff sorted-poly-list-rel-wrt-def
dest!: *set-rev-mp*[*OF - vars-polynomial-of-mset*]
dest!: *split-list*)
apply (*auto dest!:* *multi-member-split simp: list-rel-append1*
term-poly-list-rel-def eq-commute[*of - <mset ->*]
list-rel-split-right-iff list-rel-append2 list-rel-split-left-iff
dest: arg-cong[*of <mset -> <add-mset -> set-mset*])
done

lemma *check-addition-l-check-add*:

assumes $\langle (A, B) \in \text{fmap-polys-rel} \rangle$ **and** $\langle (r, r') \in \text{sorted-poly-rel } O \ \text{mset-poly-rel} \rangle$

$\langle (p, p') \in Id \rangle \langle (q, q') \in Id \rangle \langle (i, i') \in nat-rel \rangle$
 $\langle (\mathcal{V}', \mathcal{V}) \in \langle var-rel \rangle set-rel \rangle$

shows

$\langle check-addition-l\ spec\ A\ \mathcal{V}'\ p\ q\ i\ r \leq \Downarrow \{(st, b). (\neg is-cfailed\ st \longleftrightarrow b) \wedge$
 $(is-cfound\ st \longrightarrow spec = r)\} (check-add\ B\ \mathcal{V}\ p'\ q'\ i'\ r') \rangle$

proof –

have [*refine*]:

$\langle add-poly-l\ (p, q) \leq \Downarrow (sorted-poly-rel\ O\ mset-poly-rel)\ (add-poly-spec\ p'\ q') \rangle$
if $\langle (p, p') \in sorted-poly-rel\ O\ mset-poly-rel \rangle$
 $\langle (q, q') \in sorted-poly-rel\ O\ mset-poly-rel \rangle$
for $p\ p'\ q\ q'$
using *that*
by (*auto intro!*: *add-poly-l-add-poly-p'* [*THEN order-trans*] *ref-two-step'*
add-poly-p'-add-poly-spec
simp flip: conc-fun-chain)

show *?thesis*

using *assms*

unfolding *check-addition-l-def*
check-not-equal-dom-err-def **apply** –

apply (*rule order-trans*)

defer

apply (*rule ref-two-step'*)

apply (*rule check-add-alt-def*)

apply *refine-rcg*

subgoal

by (*drule sorted-poly-rel-vars-llist*)
(auto simp: set-rel-def var-rel-def br-def)

subgoal

by *auto*

subgoal

by *auto*

subgoal

by *auto*

subgoal

by *auto*

subgoal

by *auto*

subgoal

by *auto*

subgoal

by (*auto simp: weak-equality-l-def bind-RES-RETURN-eq*)

done

qed

lemma *check-del-l-check-del*:

$\langle (A, B) \in fmap-polys-rel \implies (x3, x3a) \in Id \implies check-del-l\ spec\ A\ (pac-src1\ (Del\ x3))$
 $\leq \Downarrow \{(st, b). (\neg is-cfailed\ st \longleftrightarrow b) \wedge (b \longrightarrow st = CSUCCESS)\} (check-del\ B\ (pac-src1\ (Del\ x3a))) \rangle$

unfolding *check-del-l-def check-del-def*

by (*refine-vcg lhs-step-If RETURN-SPEC-refine*)
(auto simp: fmap-rel-nat-rel-dom-m bind-RES-RETURN-eq)

lemma *check-mult-l-check-mult*:

assumes $\langle (A, B) \in fmap-polys-rel \rangle$ **and** $\langle (r, r') \in sorted-poly-rel\ O\ mset-poly-rel \rangle$ **and**
 $\langle (q, q') \in sorted-poly-rel\ O\ mset-poly-rel \rangle$

$\langle (p, p') \in Id \rangle \langle (i, i') \in nat\text{-}rel \rangle \langle (\mathcal{V}, \mathcal{V}') \in \langle var\text{-}rel \rangle set\text{-}rel \rangle$
shows
 $\langle check\text{-}mult\text{-}l\text{-}spec\ A\ \mathcal{V}\ p\ q\ i\ r \leq \Downarrow \{ (st, b). (\neg is\text{-}cfailed\ st \longleftrightarrow b) \wedge$
 $(is\text{-}cfound\ st \longrightarrow spec = r) \} (check\text{-}mult\ B\ \mathcal{V}'\ p'\ q'\ i'\ r') \rangle$
proof –
have [*refine*]:
 $\langle mult\text{-}poly\text{-}full\ p\ q \leq \Downarrow (sorted\text{-}poly\text{-}rel\ O\ mset\text{-}poly\text{-}rel) (mult\text{-}poly\text{-}spec\ p'\ q') \rangle$
if $\langle (p, p') \in sorted\text{-}poly\text{-}rel\ O\ mset\text{-}poly\text{-}rel \rangle$
 $\langle (q, q') \in sorted\text{-}poly\text{-}rel\ O\ mset\text{-}poly\text{-}rel \rangle$
for $p\ p'\ q\ q'$
using *that*
by (*auto intro!*: *mult-poly-full-mult-poly-p'*[*THEN order-trans*] *ref-two-step'*
mult-poly-p'-mult-poly-spec
simp flip: conc-fun-chain)

show *?thesis*
using *assms*
unfolding *check-mult-l-def*
check-mult-l-mult-err-def check-mult-l-dom-err-def apply –
apply (*rule order-trans*)
defer
apply (*rule ref-two-step'*)
apply (*rule check-mult-alt-def*)
apply *refine-recg*
subgoal
by (*drule sorted-poly-rel-vars-llist*) +
(fastforce simp: set-rel-def var-rel-def br-def)
subgoal
by *auto*
subgoal
by *auto*
subgoal
by *auto*
subgoal
by *auto*
subgoal
by (*auto simp: weak-equality-l-def bind-RES-RETURN-eq*)
done
qed

lemma *normalize-poly-normalize-poly-spec*:
assumes $\langle (r, t) \in unsorted\text{-}poly\text{-}rel\ O\ mset\text{-}poly\text{-}rel \rangle$
shows
 $\langle normalize\text{-}poly\ r \leq \Downarrow (sorted\text{-}poly\text{-}rel\ O\ mset\text{-}poly\text{-}rel) (normalize\text{-}poly\text{-}spec\ t) \rangle$
proof –
obtain *s* **where**
 $rs: \langle (r, s) \in unsorted\text{-}poly\text{-}rel \rangle$ **and**
 $st: \langle (s, t) \in mset\text{-}poly\text{-}rel \rangle$
using *assms* **by** *auto*
show *?thesis*
by (*rule normalize-poly-normalize-poly-p*[*THEN order-trans, OF rs*])
(use st in <auto dest!: rtranclp-normalize-poly-p-poly-of-mset
intro!: ref-two-step' RES-refine exI[of - t]
simp: normalize-poly-spec-def ideal.span-zero mset-poly-rel-def)

simp flip: conc-fun-chain)
qed

lemma *remove1-list-rel*:
 $\langle (xs, ys) \in \langle R \rangle \text{list-rel} \implies$
 $(a, b) \in R \implies$
IS-RIGHT-UNIQUE $R \implies$
IS-LEFT-UNIQUE $R \implies$
 $(\text{remove1 } a \text{ } xs, \text{remove1 } b \text{ } ys) \in \langle R \rangle \text{list-rel}$
by (*induction xs ys rule: list-rel-induct*)
(auto simp: single-valued-def IS-LEFT-UNIQUE-def)

lemma *remove1-list-rel2*:
 $\langle (xs, ys) \in \langle R \rangle \text{list-rel} \implies$
 $(a, b) \in R \implies$
 $(\bigwedge c. (a, c) \in R \implies c = b) \implies$
 $(\bigwedge c. (c, b) \in R \implies c = a) \implies$
 $(\text{remove1 } a \text{ } xs, \text{remove1 } b \text{ } ys) \in \langle R \rangle \text{list-rel}$
apply (*induction xs ys rule: list-rel-induct*)
apply (*solves <simp (no-asm)>*)
by (*smt list-rel-simp(4) remove1.simps(2)*)

lemma *remove1-sorted-poly-rel-mset-poly-rel*:
assumes
 $\langle (r, r') \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle$ **and**
 $\langle ([a], 1) \in \text{set } r \rangle$
shows
 $\langle (\text{remove1 } ([a], 1) \text{ } r, r' - \text{Var } (\varphi \text{ } a))$
 $\in \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle$

proof –
have [*simp*]: $\langle ([a], \{\#a\}) \in \text{term-poly-list-rel} \rangle$
 $\langle \bigwedge aa. ([a], aa) \in \text{term-poly-list-rel} \iff aa = \{\#a\} \rangle$
by (*auto simp: term-poly-list-rel-def*)
have *H*:
 $\langle \bigwedge aa. ([a], aa) \in \text{term-poly-list-rel} \implies aa = \{\#a\} \rangle$
 $\langle \bigwedge aa. (aa, \{\#a\}) \in \text{term-poly-list-rel} \implies aa = [a] \rangle$
by (*auto simp: single-valued-def IS-LEFT-UNIQUE-def*
term-poly-list-rel-def)

have [*simp*]: $\langle \text{Const } (1 :: \text{int}) = (1 :: \text{int } \text{mpoly}) \rangle$
by (*simp add: Const.abs-eq Const₀-one one-mpoly.abs-eq*)
have [*simp*]: $\langle \text{sorted-wrt term-order } (\text{map fst } r) \implies$
 $\text{sorted-wrt term-order } (\text{map fst } (\text{remove1 } ([a], 1) \text{ } r)) \rangle$
by (*induction r*) *auto*
have [*intro*]: $\langle \text{distinct } (\text{map fst } r) \implies \text{distinct } (\text{map fst } (\text{remove1 } x \text{ } r)) \rangle$ **for** x
by (*induction r*) (*auto dest: notin-set-remove1*)
have [*simp*]: $\langle (r, ya) \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$
 $\text{polynomial-of-mset } (\text{mset } ya) - \text{Var } (\varphi \text{ } a) =$
 $\text{polynomial-of-mset } (\text{remove1-mset } (\{\#a\}, 1) (\text{mset } ya)) \rangle$ **for** ya
using *assms*
by (*auto simp: list-rel-append1 list-rel-split-right-iff*
dest!: split-list)

show *?thesis*
using *assms*

```

apply (elim relcompEpair)
apply (rename-tac za, rule-tac b = ⟨remove1-mset ({#a#}, 1) za⟩ in relcompI)
apply (auto simp: mset-poly-rel-def sorted-poly-list-rel-wrt-def Collect-eq-comp'
  intro!: relcompI[of - ⟨remove1 ({#a#}, 1) ya⟩
    for ya :: ⟨(string multiset × int) list⟩ remove1-list-rel2 intro: H
  simp: list-mset-rel-def br-def
  dest: in-diffD)
done
qed

```

lemma *remove1-sorted-poly-rel-mset-poly-rel-minus:*

```

assumes
  ⟨(r, r') ∈ sorted-poly-rel O mset-poly-rel⟩ and
  ⟨([a], -1) ∈ set r⟩
shows
  ⟨(remove1 ([a], -1) r, r' + Var (φ a))
    ∈ sorted-poly-rel O mset-poly-rel⟩
proof -
  have [simp]: ⟨([a], {#a#}) ∈ term-poly-list-rel⟩
  ⟨∧aa. ([a], aa) ∈ term-poly-list-rel ⟷ aa = {#a#}⟩
  by (auto simp: term-poly-list-rel-def)
  have H:
  ⟨∧aa. ([a], aa) ∈ term-poly-list-rel ⟹ aa = {#a#}⟩
  ⟨∧aa. (aa, {#a#}) ∈ term-poly-list-rel ⟹ aa = [a]⟩
  by (auto simp: single-valued-def IS-LEFT-UNIQUE-def
    term-poly-list-rel-def)

```

```

have [simp]: ⟨Const (1 :: int) = (1 :: int mpoly)⟩
  by (simp add: Const.abs-eq Const0-one one-mpoly.abs-eq)
have [simp]: ⟨sorted-wrt term-order (map fst r) ⟹
  sorted-wrt term-order (map fst (remove1 ([a], -1) r))⟩
  by (induction r) auto
have [intro]: ⟨distinct (map fst r) ⟹ distinct (map fst (remove1 x r))⟩ for x
  apply (induction r) apply auto
  by (meson img-fst in-set-remove1D)
have [simp]: ⟨(r, ya) ∈ ⟨term-poly-list-rel ×r int-rel⟩list-rel ⟹
  polynomial-of-mset (mset ya) + Var (φ a) =
  polynomial-of-mset (remove1-mset ({#a#}, -1) (mset ya))⟩ for ya
using assms
  by (auto simp: list-rel-append1 list-rel-split-right-iff
    dest!: split-list)

```

show *?thesis*

```

using assms
apply (elim relcompEpair)
apply (rename-tac za, rule-tac b = ⟨remove1-mset ({#a#}, -1) za⟩ in relcompI)
by (auto simp: mset-poly-rel-def sorted-poly-list-rel-wrt-def Collect-eq-comp'
  dest: in-diffD
  intro!: relcompI[of - ⟨remove1 ({#a#}, -1) ya⟩
    for ya :: ⟨(string multiset × int) list⟩ remove1-list-rel2 intro: H
  simp: list-mset-rel-def br-def)
qed

```

lemma *insert-var-rel-set-rel:*

$\langle \mathcal{V}, \mathcal{V}' \rangle \in \langle \text{var-rel} \rangle \text{set-rel} \implies$
 $\langle yb, x2 \rangle \in \text{var-rel} \implies$
 $\langle \text{insert } yb \ \mathcal{V}, \text{insert } x2 \ \mathcal{V}' \rangle \in \langle \text{var-rel} \rangle \text{set-rel}$
by (*auto simp: var-rel-def set-rel-def*)

lemma *var-rel-set-rel-iff*:

$\langle \mathcal{V}, \mathcal{V}' \rangle \in \langle \text{var-rel} \rangle \text{set-rel} \implies$
 $\langle yb, x2 \rangle \in \text{var-rel} \implies$
 $yb \in \mathcal{V} \longleftrightarrow x2 \in \mathcal{V}'$
using φ -inj inj-eq **by** (*fastforce simp: var-rel-def set-rel-def br-def*)

lemma *check-extension-l-check-extension*:

assumes $\langle A, B \rangle \in \text{fmap-polys-rel}$ **and** $\langle r, r' \rangle \in \text{sorted-poly-rel } O \text{ mset-poly-rel}$ **and**
 $\langle i, i' \rangle \in \text{nat-rel}$ $\langle \mathcal{V}, \mathcal{V}' \rangle \in \langle \text{var-rel} \rangle \text{set-rel}$ $\langle x, x' \rangle \in \text{var-rel}$
shows
 $\langle \text{check-extension-l spec } A \ \mathcal{V} \ i \ x \ r \leq$
 $\Downarrow \{((st), (b)).$
 $(\neg \text{is-cfailed } st \longleftrightarrow b) \wedge$
 $(\text{is-cfound } st \longrightarrow \text{spec} = r)\} (\text{check-extension } B \ \mathcal{V}' \ i' \ x' \ r') \rangle$

proof –

have $\langle x' = \varphi \ x \rangle$
using *assms(5)* **by** (*auto simp: var-rel-def br-def*)
have [*refine*]:
 $\langle \text{mult-poly-full } p \ q \leq \Downarrow (\text{sorted-poly-rel } O \text{ mset-poly-rel}) (\text{mult-poly-spec } p' \ q') \rangle$
if $\langle p, p' \rangle \in \text{sorted-poly-rel } O \text{ mset-poly-rel}$
 $\langle q, q' \rangle \in \text{sorted-poly-rel } O \text{ mset-poly-rel}$
for $p \ p' \ q \ q'$
using *that*
by (*auto intro!: mult-poly-full-mult-poly-p'[THEN order-trans] ref-two-step'*
 $\text{mult-poly-p'-mult-poly-spec}$
 $\text{simp flip: conc-fun-chain}$)
have [*refine*]:
 $\langle \text{add-poly-l } (p, q) \leq \Downarrow (\text{sorted-poly-rel } O \text{ mset-poly-rel}) (\text{add-poly-spec } p' \ q') \rangle$
if $\langle p, p' \rangle \in \text{sorted-poly-rel } O \text{ mset-poly-rel}$
 $\langle q, q' \rangle \in \text{sorted-poly-rel } O \text{ mset-poly-rel}$
for $p \ p' \ q \ q'$
using *that*
by (*auto intro!: add-poly-l-add-poly-p'[THEN order-trans] ref-two-step'*
 $\text{add-poly-p'-add-poly-spec}$
 $\text{simp flip: conc-fun-chain}$)

have [*simp*]: $\langle l, l' \rangle \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$
 $(\text{map } (\lambda(a, b). (a, - b)) \ l, \text{map } (\lambda(a, b). (a, - b)) \ l')$
 $\in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel}$ **for** $l \ l'$
by (*induction l l' rule: list-rel-induct*)
(auto simp: list-mset-rel-def br-def)

have [*intro!*]:

$\langle x2c, za \rangle \in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel} \implies$
 $(\text{map } (\lambda(a, b). (a, - b)) \ x2c,$
 $\{\# \text{case } x \text{ of } (a, b) \Rightarrow (a, - b). x \in \# \text{ za} \#\})$
 $\in \langle \text{term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel } O \text{ list-mset-rel}$ **for** $x2c \ za$
apply (*auto*)
subgoal **for** y

```

apply (induction x2c y rule: list-rel-induct)
apply (auto simp: list-mset-rel-def br-def)
apply (rename-tac a ba aa l l', rule-tac b = ⟨(aa, - ba) # map (λ(a, b). (a, - b)) l'⟩ in relcompI)
by auto
done
have [simp]: ⟨(λx. fst (case x of (a, b) ⇒ (a, - b))) = fst⟩
by (auto intro: ext)

have uminus: ⟨(x2c, x2a) ∈ sorted-poly-rel O mset-poly-rel ⇒
  (map (λ(a, b). (a, - b)) x2c,
   - x2a)
  ∈ sorted-poly-rel O mset-poly-rel⟩ for x2c x2a x1c x1a
apply (clarsimp simp: sorted-poly-list-rel-wrt-def
  mset-poly-rel-def)
apply (rule-tac b = ⟨(λ(a, b). (a, - b)) '# za⟩ in relcompI)
by (auto simp: sorted-poly-list-rel-wrt-def
  mset-poly-rel-def comp-def polynomial-of-mset-uminus)
have [simp]: ⟨([], 0) ∈ sorted-poly-rel O mset-poly-rel⟩
by (auto simp: sorted-poly-list-rel-wrt-def
  mset-poly-rel-def list-mset-rel-def br-def
  intro!: relcompI[of - ⟨{#}⟩])
show ?thesis
unfolding check-extension-l-def
  check-extension-l-dom-err-def
  check-extension-l-no-new-var-err-def
  check-extension-l-new-var-multiple-err-def
  check-extension-l-side-cond-err-def
apply (rule order-trans)
defer
apply (rule ref-two-step')
apply (rule check-extension-alt-def)
apply (refine-vcg )
subgoal using assms(1,3,4,5)
  by (auto simp: var-rel-set-rel-iff)
subgoal using assms(1,3,4,5)
  by (auto simp: var-rel-set-rel-iff)
subgoal by auto
subgoal by auto
apply (subst ⟨x' = φ x⟩, rule remove1-sorted-poly-rel-mset-poly-rel-minus)
subgoal using assms by auto
subgoal using assms by auto
subgoal using sorted-poly-rel-vars-llist[of ⟨r⟩ ⟨r'⟩] assms
  by (force simp: set-rel-def var-rel-def br-def
  dest!: sorted-poly-rel-vars-llist)
subgoal by auto
subgoal by auto
subgoal using assms by auto
subgoal using assms by auto
apply (rule uminus)
subgoal using assms by auto
subgoal using assms by auto
subgoal using assms by auto
subgoal using assms by auto
subgoal using assms by auto
done

```

qed

lemma *full-normalize-poly-diff-ideal*:

fixes *dom*

assumes $\langle (p, p') \in \text{fully-unsorted-poly-rel } O \text{ mset-poly-rel} \rangle$

shows

$\langle \text{full-normalize-poly } p$
 $\leq \Downarrow (\text{sorted-poly-rel } O \text{ mset-poly-rel})$
 $(\text{normalize-poly-spec } p') \rangle$

proof –

obtain *q* **where**

$pq: \langle (p, q) \in \text{fully-unsorted-poly-rel} \rangle$ **and** $qp': \langle (q, p') \in \text{mset-poly-rel} \rangle$

using *assms* **by** *auto*

show *?thesis*

unfolding *normalize-poly-spec-def*

apply (*rule full-normalize-poly-normalize-poly-p* [*THEN order-trans*])

apply (*rule pq*)

unfolding *conc-fun-chain[symmetric]*

by (*rule ref-two-step'*, *rule RES-refine*)

(*use qp' in* $\langle \text{auto dest!}: \text{rtranclp-normalize-poly-p-poly-of-mset}$
 $\text{simp}: \text{mset-poly-rel-def ideal.span-zero} \rangle$)

qed

lemma *insort-key-rel-decomp*:

$\langle \exists ys zs. xs = ys @ zs \wedge \text{insort-key-rel } R \ x \ xs = ys @ x \# zs \rangle$

apply (*induction xs*)

subgoal **by** *auto*

subgoal **for** *a xs*

by (*force intro: exI[of -* $\langle a \# - \rangle$)

done

lemma *list-rel-append-same-length*:

$\langle \text{length } xs = \text{length } xs' \implies (xs @ ys, xs' @ ys') \in \langle R \rangle \text{list-rel} \longleftrightarrow (xs, xs') \in \langle R \rangle \text{list-rel} \wedge (ys, ys') \in \langle R \rangle \text{list-rel} \rangle$

by (*auto simp: list-rel-def list-all2-append2 dest: list-all2-lengthD*)

lemma *term-poly-list-rel-list-relD*: $\langle (ys, cs) \in \langle \text{term-poly-list-rel } \times_r \text{ int-rel} \rangle \text{list-rel} \implies$

$cs = \text{map } (\lambda(a, y). (\text{mset } a, y)) \ ys \rangle$

by (*induction ys arbitrary: cs*)

(*auto simp: term-poly-list-rel-def list-rel-def list-all2-append list-all2-Cons1 list-all2-Cons2*)

lemma *term-poly-list-rel-single*: $\langle ([x32], \{\#x32\#}) \in \text{term-poly-list-rel} \rangle$

by (*auto simp: term-poly-list-rel-def*)

lemma *unsorted-poly-rel-list-rel-list-rel-uminus*:

$\langle \text{map } (\lambda(a, b). (a, - b)) \ r, yc \rangle$
 $\in \langle \text{unsorted-term-poly-list-rel } \times_r \text{ int-rel} \rangle \text{list-rel} \implies$
 $(r, \text{map } (\lambda(a, b). (a, - b)) \ yc)$
 $\in \langle \text{unsorted-term-poly-list-rel } \times_r \text{ int-rel} \rangle \text{list-rel} \rangle$

by (*induction r arbitrary: yc*)

(*auto simp: elim!: list-relE3*)

lemma *mset-poly-rel-minus*: $\langle (\{\#(a, b)\#}, v') \in \text{mset-poly-rel} \implies$

$(\text{mset } yc, r') \in \text{mset-poly-rel} \implies$

(r, yc)
 $\in \langle \text{unsorted-term-poly-list-rel} \times_r \text{int-rel} \rangle \text{list-rel} \implies$
 $(\text{add-mset } (a, b) \text{ (mset } yc),$
 $v' + r')$
 $\in \text{mset-poly-rel} \rangle$
by (induction r arbitrary: r')
 (auto simp: mset-poly-rel-def polynomial-of-mset-uminus)

lemma *fully-unsorted-poly-rel-diff*:

$\langle ([v], v') \in \text{fully-unsorted-poly-rel} \ O \ \text{mset-poly-rel} \implies$
 $(r, r') \in \text{fully-unsorted-poly-rel} \ O \ \text{mset-poly-rel} \implies$
 $(v \# r,$
 $v' + r')$
 $\in \text{fully-unsorted-poly-rel} \ O \ \text{mset-poly-rel} \rangle$

apply *auto*

apply (rule-tac $b = \langle y + ya \rangle$ **in** *relcompI*)

apply (auto simp: fully-unsorted-poly-list-rel-def list-mset-rel-def br-def)

apply (rule-tac $b = \langle yb \ @ \ yc \rangle$ **in** *relcompI*)

apply (auto elim!: list-relE3 simp: unsorted-poly-rel-list-rel-list-rel-uminus mset-poly-rel-minus)

done

lemma *PAC-checker-l-step-PAC-checker-step*:

assumes

$\langle (Ast, Bst) \in \text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel} \rangle$ **and**

$\langle (st, st') \in \text{pac-step-rel} \rangle$ **and**

spec: $\langle (\text{spec}, \text{spec}') \in \text{sorted-poly-rel} \ O \ \text{mset-poly-rel} \rangle$

shows

$\langle \text{PAC-checker-l-step spec Ast st} \leq \Downarrow (\text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel})$
 $(\text{PAC-checker-step spec}' Bst st') \rangle$

proof –

obtain $A \ \mathcal{V} \ \text{cst} \ B \ \mathcal{V}' \ \text{cst}'$ **where**

Ast : $\langle Ast = (\text{cst}, \mathcal{V}, A) \rangle$ **and**

Bst : $\langle Bst = (\text{cst}', \mathcal{V}', B) \rangle$ **and**

$\mathcal{V}[\text{intro}]$: $\langle (\mathcal{V}, \mathcal{V}') \in \langle \text{var-rel} \rangle \text{set-rel} \rangle$ **and**

AB : $\langle (A, B) \in \text{fmap-polys-rel} \rangle$

$\langle (\text{cst}, \text{cst}') \in \text{code-status-status-rel} \rangle$

using *assms(1)*

by (cases Ast ; cases Bst ; *auto*)

have [*refine*]: $\langle (r, ra) \in \text{sorted-poly-rel} \ O \ \text{mset-poly-rel} \implies$

(*eqa*, *eqaa*)

$\in \{(st, b). (\neg \text{is-cfailed } st \longleftrightarrow b) \wedge (\text{is-cfound } st \longrightarrow \text{spec} = r)\} \implies$

RETURN eqa

$\leq \Downarrow \text{code-status-status-rel}$

(*SPEC*

$(\lambda st'. (\neg \text{is-failed } st' \wedge$

is-found } st' \longrightarrow

$ra - \text{spec}' \in \text{More-Modules.ideal polynomial-bool})) \rangle$

for $r \ ra \ eqa \ eqaa$

using *spec*

by (cases *eqa*)

(*auto intro!*: *RETURN-RES-refine dest!*: *sorted-poly-list-relD*

simp: *mset-poly-rel-def ideal.span-zero*)

have [*simp*]: $\langle (\text{eqa}, st'a) \in \text{code-status-status-rel} \implies$

(*merge-cstatus cst eqa*, *merge-status cst' st'a*)

$\in \text{code-status-status-rel} \rangle$ **for** $eqa \ st'a$

```

using AB
by (cases eqa; cases st'a)
  (auto simp: code-status-status-rel-def)
have [simp]:  $\langle (\text{merge-cstatus } cst \text{ CSUCCESS}, cst') \in \text{code-status-status-rel} \rangle$ 
using AB
by (cases st)
  (auto simp: code-status-status-rel-def)
have [simp]:  $\langle (x32, x32a) \in \text{var-rel} \implies$ 
  ( $[[x32], -1 :: \text{int}], -\text{Var } x32a) \in \text{fully-unsorted-poly-rel } O \text{ mset-poly-rel} \rangle$  for  $x32 \ x32a$ 
by (auto simp: mset-poly-rel-def fully-unsorted-poly-list-rel-def list-mset-rel-def br-def
  unsorted-term-poly-list-rel-def var-rel-def Const-1-eq-1
  intro!: relcompI[of -  $\langle \#(\{\#x32\}, -1 :: \text{int})\# \rangle$ ])
  relcompI[of -  $\langle [\{\#x32\}, -1] \rangle$ ])
have H3:  $\langle p - \text{Var } a = (-\text{Var } a) + p \rangle$  for  $p :: \langle \text{int } \text{mpoly} \rangle$  and  $a$ 
by auto
show ?thesis
using assms(2)
unfolding PAC-checker-l-step-def PAC-checker-step-def Ast Bst prod.case
apply (cases st; cases st'; simp only: p2rel-def pac-step.case
  pac-step-rel-raw-def mem-Collect-eq prod.case pac-step-rel-raw.simps)
subgoal
apply (refine-rcg normalize-poly-normalize-poly-spec
  check-mult-l-check-mult check-addition-l-check-add
  full-normalize-poly-diff-ideal)
subgoal using AB by auto
subgoal using AB by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by (auto intro: V)
apply assumption+
subgoal
by (auto simp: code-status-status-rel-def)
subgoal
by (auto intro!: fmap-rel-fmupd-fmap-rel
  fmap-rel-fmdrop-fmap-rel AB)
subgoal using AB by auto
done
subgoal
apply (refine-rcg normalize-poly-normalize-poly-spec
  check-mult-l-check-mult check-addition-l-check-add
  full-normalize-poly-diff-ideal[unfolded normalize-poly-spec-def[symmetric]])
subgoal using AB by auto
subgoal using AB by auto
subgoal using AB by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
apply assumption+
subgoal
by (auto simp: code-status-status-rel-def)
subgoal
by (auto intro!: fmap-rel-fmupd-fmap-rel
  fmap-rel-fmdrop-fmap-rel AB)
subgoal using AB by auto

```



```

done
subgoal
  apply (refine-rcg full-normalize-poly-diff-ideal
    check-extension-l-check-extension)
  subgoal using AB by (auto intro!: fully-unsorted-poly-rel-diff[of - <- Var - :: int mpoly>, unfolded
H3[symmetric]] simp: comp-def case-prod-beta)
  subgoal using AB by auto
  subgoal using AB by auto
  subgoal by auto
  subgoal by auto
  subgoal
    by (auto simp: code-status-status-rel-def)
  subgoal
    by (auto simp: AB
      intro!: fmap-rel-fmupd-fmap-rel insert-var-rel-set-rel)
  subgoal
    by (auto simp: code-status-status-rel-def AB
      code-status.is-cfailed-def)
done
subgoal
  apply (refine-rcg normalize-poly-normalize-poly-spec
    check-del-l-check-del check-addition-l-check-add
    full-normalize-poly-diff-ideal[unfolded normalize-poly-spec-def[symmetric]])
  subgoal using AB by auto
  subgoal using AB by auto
  subgoal
    by (auto intro!: fmap-rel-fmupd-fmap-rel
      fmap-rel-fmdrop-fmap-rel code-status-status-rel-def AB)
  subgoal
    by (auto intro!: fmap-rel-fmupd-fmap-rel
      fmap-rel-fmdrop-fmap-rel AB)
done
done
qed

```

lemma *code-status-status-rel-discrim-iff*:

```

<(x1a, x1c) ∈ code-status-status-rel ⇒ is-cfailed x1a ↔ is-failed x1c>
<(x1a, x1c) ∈ code-status-status-rel ⇒ is-cfound x1a ↔ is-found x1c>
by (cases x1a; cases x1c; auto; fail)+

```

lemma *PAC-checker-l-PAC-checker*:

```

assumes
  <(A, B) ∈ <var-rel>set-rel ×r fmap-polys-rel> and
  <(st, st') ∈ <pac-step-rel>list-rel> and
  <(spec, spec') ∈ sorted-poly-rel O mset-poly-rel> and
  <(b, b') ∈ code-status-status-rel>
shows
  <PAC-checker-l spec A b st ≤ ↓ (code-status-status-rel ×r <var-rel>set-rel ×r fmap-polys-rel) (PAC-checker
spec' B b' st')>
proof -
  have [refine0]: <(((b, A), st), (b', B), st') ∈ ((code-status-status-rel ×r <var-rel>set-rel ×r fmap-polys-rel)
×r <pac-step-rel>list-rel)>
  using assms by (auto simp: code-status-status-rel-def)
  show ?thesis
  using assms

```

```

unfolding PAC-checker-l-def PAC-checker-def
apply (refine-recq PAC-checker-l-step-PAC-checker-step
  WHILEIT-refine[where R = ⟨((bool-rel ×r ⟨var-rel⟩set-rel ×r fmap-polys-rel) ×r ⟨pac-step-rel⟩list-rel)⟩])
subgoal by (auto simp: code-status-status-rel-discrim-iff)
subgoal by auto
subgoal by (auto simp: neq-Nil-conv)
subgoal by (auto simp: neq-Nil-conv intro!: param-nth)
subgoal by (auto simp: neq-Nil-conv)
subgoal by auto
done
qed

```

end

```

lemma less-than-char-of-char[code-unfold]:
  ⟨(x, y) ∈ less-than-char ⟷ (of-char x :: nat) < of-char y⟩
by (auto simp: less-than-char-def less-char-def)

```

```

lemmas [code] =
  add-poly-l'.simps[unfolded var-order-rel-def]

```

export-code add-poly-l' **in** SML **module-name** test

```

definition full-checker-l
  :: ⟨llist-polynomial ⇒ (nat, llist-polynomial) fmap ⇒ (-, string, nat) pac-step list ⇒
    (string code-status × -) nres⟩
where
  ⟨full-checker-l spec A st = do {
    spec' ← full-normalize-poly spec;
    (b, V, A) ← remap-polys-l spec' {} A;
    if is-cfailed b
    then RETURN (b, V, A)
    else do {
      let V = V ∪ vars-llist spec;
      PAC-checker-l spec' (V, A) b st
    }
  }⟩

```

```

context poly-embed
begin

```

```

term normalize-poly-spec
thm full-normalize-poly-diff-ideal[unfolded normalize-poly-spec-def[symmetric]]
abbreviation unsorted-fmap-polys-rel where
  ⟨unsorted-fmap-polys-rel ≡ ⟨nat-rel, fully-unsorted-poly-rel O mset-poly-rel⟩fmap-rel⟩

```

```

lemma full-checker-l-full-checker:
assumes
  ⟨(A, B) ∈ unsorted-fmap-polys-rel⟩ and
  ⟨(st, st') ∈ ⟨pac-step-rel⟩list-rel⟩ and
  ⟨(spec, spec') ∈ fully-unsorted-poly-rel O mset-poly-rel⟩
shows

```

$\langle \text{full-checker-l spec } A \text{ st } \leq \Downarrow (\text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel}) (\text{full-checker spec}' B \text{ st}') \rangle$

proof –

have [refine]:

$\langle (\text{spec}, \text{spec}') \in \text{sorted-poly-rel } O \text{ mset-poly-rel} \implies$

$(\mathcal{V}, \mathcal{V}') \in \langle \text{var-rel} \rangle \text{set-rel} \implies$

$\text{remap-polys-l spec } \mathcal{V} A \leq \Downarrow (\text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel})$

$(\text{remap-polys-change-all spec}' \mathcal{V}' B) \rangle$ **for** $\text{spec spec}' \mathcal{V} \mathcal{V}'$

apply (rule *remap-polys-l-remap-polys*[*THEN order-trans, OF assms(1)*])

apply *assumption+*

apply (rule *ref-two-step*[*OF order.refl*])

apply(rule *remap-polys-spec*[*THEN order-trans*])

by (rule *remap-polys-polynomial-bool-remap-polys-change-all*)

show ?thesis

unfolding *full-checker-l-def full-checker-def*

apply (*refine-rcg remap-polys-l-remap-polys*

full-normalize-poly-diff-ideal[*unfolded normalize-poly-spec-def*[*symmetric*]]

PAC-checker-l-PAC-checker)

subgoal

using *assms(3)* .

subgoal by *auto*

subgoal by (*auto simp: is-cfailed-def is-failed-def*)

subgoal by *auto*

apply (rule *fully-unsorted-poly-rel-extend-vars*)

subgoal using *assms(3)* .

subgoal by *auto*

subgoal by *auto*

subgoal

using *assms(2)* **by** (*auto simp: p2rel-def*)

subgoal by *auto*

done

qed

lemma *full-checker-l-full-checker'*:

$\langle (\text{uncurry2 full-checker-l}, \text{uncurry2 full-checker}) \in$

$((\text{fully-unsorted-poly-rel } O \text{ mset-poly-rel}) \times_r \text{unsorted-fmap-polys-rel}) \times_r \langle \text{pac-step-rel} \rangle \text{list-rel} \rightarrow_f$

$\langle (\text{code-status-status-rel} \times_r \langle \text{var-rel} \rangle \text{set-rel} \times_r \text{fmap-polys-rel}) \rangle \text{nres-rel} \rangle$

apply (*intro frefI nres-relI*)

using *full-checker-l-full-checker* **by** *force*

end

definition *remap-polys-l2* :: $\langle \text{l-list-polynomial} \Rightarrow \text{string set} \Rightarrow (\text{nat}, \text{l-list-polynomial}) \text{fmap} \Rightarrow - \text{nres} \rangle$

where

$\langle \text{remap-polys-l2 spec} = (\lambda \mathcal{V} A. \text{do}\{$

$n \leftarrow \text{upper-bound-on-dom } A;$

$b \leftarrow \text{RETURN } (n \geq 2^{64});$

if b

then do {

$c \leftarrow \text{remap-polys-l-dom-err};$

$\text{RETURN } (\text{error-msg } (0 :: \text{nat}) c, \mathcal{V}, \text{fmempty})$

}

else do {

```

(b,  $\mathcal{V}$ , A) ← nfoldli ([0..<n]) (λ-. True)
(λi (b,  $\mathcal{V}$ , A')
  if i ∈# dom-m A
  then do {
    ASSERT(fmlookup A i ≠ None);
    p ← full-normalize-poly (the (fmlookup A i));
    eq ← weak-equality-l p spec;
     $\mathcal{V}$  ← RETURN ( $\mathcal{V} \cup \text{vars-llist (the (fmlookup A i))}$ );
    RETURN(b ∨ eq,  $\mathcal{V}$ , fmupd i p A')
  } else RETURN (b,  $\mathcal{V}$ , A')
)
(False,  $\mathcal{V}$ , fmempty);
RETURN (if b then CFOUND else CSUCCESS,  $\mathcal{V}$ , A)
}
}

```

lemma *remap-polys-l2-remap-polys-l:*

⟨remap-polys-l2 spec \mathcal{V} A ≤ ↓ Id (remap-polys-l spec \mathcal{V} A)⟩

proof –

have [refine]: ⟨(A, A') ∈ Id ⇒ upper-bound-on-dom A
 ≤ ↓ {(n, dom). dom = set [0..<n]} (SPEC (λdom. set-mset (dom-m A') ⊆ dom ∧ finite dom))⟩ **for**
 A A'

unfolding upper-bound-on-dom-def
apply (rule RES-refine)
apply (auto simp: upper-bound-on-dom-def)
done

have 1: ⟨inj-on id dom⟩ **for** dom
by auto

have 2: ⟨x ∈# dom-m A ⇒
 x' ∈# dom-m A' ⇒
 (x, x') ∈ nat-rel ⇒
 (A, A') ∈ Id ⇒
 full-normalize-poly (the (fmlookup A x))
 ≤ ↓ Id
 (full-normalize-poly (the (fmlookup A' x'))))⟩
for A A' x x'
by (auto)

have 3: ⟨(n, dom) ∈ {(n, dom). dom = set [0..<n]} ⇒
 ([0..<n], dom) ∈ ⟨nat-rel⟩list-set-rel⟩ **for** n dom

by (auto simp: list-set-rel-def br-def)

have 4: ⟨(p,q) ∈ Id ⇒
 weak-equality-l p spec ≤ ↓Id (weak-equality-l q spec)⟩ **for** p q spec
by auto

have 6: ⟨a = b ⇒ (a, b) ∈ Id⟩ **for** a b
by auto

show ?thesis

unfolding remap-polys-l2-def remap-polys-l-def
apply (refine-rcg LFO-refine[**where** R= ⟨Id ×_r ⟨Id⟩set-rel ×_r Id⟩])
subgoal **by** auto
subgoal **by** auto
subgoal **by** auto
apply (rule 3)
subgoal **by** auto
subgoal **by** (simp add: in-dom-m-lookup-iff)

```

subgoal by (simp add: in-dom-m-lookup-iff)
apply (rule 2)
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
apply (rule 4; assumption)
apply (rule 6)
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
done
qed

end

theory PAC-Checker-Relation
imports PAC-Checker WB-Sort Native-Word.Uint64
begin

```

10 Various Refinement Relations

When writing this, it was not possible to share the definition with the IsaSAT version.

definition *uint64-nat-rel* :: $(uint64 \times nat)$ set **where**
 $\langle uint64\text{-}nat\text{-}rel = br\ nat\text{-}of\text{-}uint64\ (\lambda\cdot. True) \rangle$

abbreviation *uint64-nat-assn* **where**
 $\langle uint64\text{-}nat\text{-}assn \equiv pure\ uint64\text{-}nat\text{-}rel \rangle$

instantiation *uint32* :: hashable
begin

definition *hashcode-uint32* :: $\langle uint32 \Rightarrow uint32 \rangle$ **where**
 $\langle hashcode\text{-}uint32\ n = n \rangle$

definition *def-hashmap-size-uint32* :: $\langle uint32\ itself \Rightarrow nat \rangle$ **where**
 $\langle def\text{-}hashmap\text{-}size\text{-}uint32 = (\lambda\cdot. 16) \rangle$
— same as *nat*

instance
by *standard* (simp add: def-hashmap-size-uint32-def)
end

instantiation *uint64* :: hashable
begin

context
includes *bit-operations-syntax*
begin

definition *hashcode-uint64* :: $\langle uint64 \Rightarrow uint32 \rangle$ **where**
 $\langle hashcode\text{-}uint64\ n = (uint32\text{-}of\text{-}nat\ (nat\text{-}of\text{-}uint64\ ((n)\ AND\ ((2 :: uint64) \wedge^{32} - 1)))) \rangle$

end

definition *def-hashmap-size-uint64* :: $\langle \text{uint64} \text{ itself} \Rightarrow \text{nat} \rangle$ **where**
 $\langle \text{def-hashmap-size-uint64} = (\lambda-. 16) \rangle$
— same as *nat*

instance
by *standard* (*simp add: def-hashmap-size-uint64-def*)
end

lemma *word-nat-of-uint64-Rep-inject*[*simp*]: $\langle \text{nat-of-uint64 } ai = \text{nat-of-uint64 } bi \iff ai = bi \rangle$
by *transfer* (*simp add: word-unat-eq-iff*)

instance *uint64* :: *heap*
by *standard* (*auto simp: inj-def exI[of - nat-of-uint64]*)

instance *uint64* :: *semiring-numeral*
by *standard*

lemma *nat-of-uint64-012*[*simp*]: $\langle \text{nat-of-uint64 } 0 = 0 \rangle \langle \text{nat-of-uint64 } 2 = 2 \rangle \langle \text{nat-of-uint64 } 1 = 1 \rangle$
by (*simp-all add: nat-of-uint64.rep-eq zero-uint64.rep-eq one-uint64.rep-eq*)

definition *uint64-of-nat-conv* **where**
[*simp*]: $\langle \text{uint64-of-nat-conv } (x :: \text{nat}) = x \rangle$

lemma *less-upper-bintrunc-id*: $\langle n < 2^b \implies n \geq 0 \implies \text{take-bit } b \ n = n \rangle$ **for** $n :: \text{int}$
by (*rule take-bit-int-eq-self*)

lemma *nat-of-uint64-uint64-of-nat-id*: $\langle n < 2^{64} \implies \text{nat-of-uint64 } (\text{uint64-of-nat } n) = n \rangle$
by *transfer* (*simp add: take-bit-nat-eq-self unsigned-of-nat*)

lemma [*sepref-fr-rules*]:
 $\langle (\text{return } o \ \text{uint64-of-nat}, \text{RETURN } o \ \text{uint64-of-nat-conv}) \in [\lambda a. a < 2^{64}]_a \ \text{nat-assn}^k \rightarrow \text{uint64-nat-assn} \rangle$
by *sepref-to-hoare*
(*sep-auto simp: uint64-nat-rel-def br-def nat-of-uint64-uint64-of-nat-id*)

definition *string-rel* :: $\langle (\text{String.literal} \times \text{string}) \ \text{set} \rangle$ **where**
 $\langle \text{string-rel} = \{(x, y). y = \text{String.explode } x\} \rangle$

abbreviation *string-assn* :: $\langle \text{string} \Rightarrow \text{String.literal} \Rightarrow \text{assn} \rangle$ **where**
 $\langle \text{string-assn} \equiv \text{pure string-rel} \rangle$

lemma *eq-string-eq*:
 $\langle ((=), (=)) \in \text{string-rel} \rightarrow \text{string-rel} \rightarrow \text{bool-rel} \rangle$
by (*auto intro!: frefI simp: string-rel-def String.less-literal-def less-than-char-def rel2p-def literal.explode-inject*)

lemmas *eq-string-eq-hnr* =
eq-string-eq[*sepref-import-param*]

definition *string2-rel* :: $\langle (\text{string} \times \text{string}) \ \text{set} \rangle$ **where**
 $\langle \text{string2-rel} \equiv \langle \text{Id} \rangle \text{list-rel} \rangle$

abbreviation *string2-assn* :: $\langle \text{string} \Rightarrow \text{string} \Rightarrow \text{assn} \rangle$ **where**
 $\langle \text{string2-assn} \equiv \text{pure string2-rel} \rangle$

abbreviation *monom-rel* **where**

⟨*monom-rel* ≡ ⟨*string-rel*⟩*list-rel*⟩

abbreviation *monom-assn* **where**

⟨*monom-assn* ≡ *list-assn string-assn*⟩

abbreviation *monomial-rel* **where**

⟨*monomial-rel* ≡ *monom-rel* ×_r *int-rel*⟩

abbreviation *monomial-assn* **where**

⟨*monomial-assn* ≡ *monom-assn* ×_a *int-assn*⟩

abbreviation *poly-rel* **where**

⟨*poly-rel* ≡ ⟨*monomial-rel*⟩*list-rel*⟩

abbreviation *poly-assn* **where**

⟨*poly-assn* ≡ *list-assn monomial-assn*⟩

lemma *poly-assn-alt-def*:

⟨*poly-assn* = *pure poly-rel*⟩

by (*simp add: list-assn-pure-conv*)

abbreviation *polys-assn* **where**

⟨*polys-assn* ≡ *hm-fmap-assn uint64-nat-assn poly-assn*⟩

lemma *string-rel-string-assn*:

⟨(↑ ((*c*, *a*) ∈ *string-rel*)) = *string-assn a c*⟩

by (*auto simp: pure-app-eq*)

lemma *single-valued-string-rel*:

⟨*single-valued string-rel*⟩

by (*auto simp: single-valued-def string-rel-def*)

lemma *IS-LEFT-UNIQUE-string-rel*:

⟨*IS-LEFT-UNIQUE string-rel*⟩

by (*auto simp: IS-LEFT-UNIQUE-def single-valued-def string-rel-def*
literal.explode-inject)

lemma *IS-RIGHT-UNIQUE-string-rel*:

⟨*IS-RIGHT-UNIQUE string-rel*⟩

by (*auto simp: single-valued-def string-rel-def*
literal.explode-inject)

lemma *single-valued-monom-rel*: ⟨*single-valued monom-rel*⟩

by (*rule list-rel-sv*)

(*auto intro!: freqI simp: string-rel-def*
rel2p-def single-valued-def p2rel-def)

lemma *single-valued-monomial-rel*:

⟨*single-valued monomial-rel*⟩

using *single-valued-monom-rel*

by (*auto intro!: freqI simp:*
rel2p-def single-valued-def p2rel-def)

lemma *single-valued-monom-rel'*: ⟨*IS-LEFT-UNIQUE monom-rel*⟩

unfolding *IS-LEFT-UNIQUE-def inv-list-rel-eq string2-rel-def*
by (*rule list-rel-sv*) +
(*auto intro!*: *freqI simp: string-rel-def*
rel2p-def single-valued-def p2rel-def literal.explode-inject)

lemma *single-valued-monomial-rel'*:
 \langle *IS-LEFT-UNIQUE monomial-rel* \rangle
using *single-valued-monom-rel'*
unfolding *IS-LEFT-UNIQUE-def inv-list-rel-eq*
by (*auto intro!*: *freqI simp:*
rel2p-def single-valued-def p2rel-def)

lemma [*safe-constraint-rules*]:
 \langle *Sepref-Constraints.CONSTRAINT single-valued string-rel* \rangle
 \langle *Sepref-Constraints.CONSTRAINT IS-LEFT-UNIQUE string-rel* \rangle
by (*auto simp: CONSTRAINT-def single-valued-def*
string-rel-def IS-LEFT-UNIQUE-def literal.explode-inject)

lemma *eq-string-monom-hnr*[*sepref-fr-rules*]:
 \langle (*uncurry (return oo (=)), uncurry (RETURN oo (=)))* \in *monom-assn^k *_a monom-assn^k \rightarrow_a bool-assn* \rangle
using *single-valued-monom-rel' single-valued-monom-rel*
unfolding *list-assn-pure-conv*
by *sepref-to-hoare*
(*sep-auto simp: list-assn-pure-conv string-rel-string-assn*
single-valued-def IS-LEFT-UNIQUE-def
dest!: mod-starD
simp flip: inv-list-rel-eq)

definition *term-order-rel' where*
[*simp*]: \langle *term-order-rel' x y = ((x, y) \in term-order-rel)* \rangle

lemma *term-order-rel*[*def-pat-rules*]:
 \langle (\in)\$(*x,y*)\$*term-order-rel* \equiv *term-order-rel'*\$(*x*)\$(*y*) \rangle
by *auto*

lemma *term-order-rel-alt-def*:
 \langle *term-order-rel = lexord (p2rel char.lexordp)* \rangle
by (*auto simp: p2rel-def char.lexordp-conv-lexord var-order-rel-def intro!*: *arg-cong[of - - lexord]*)

instantiation *char :: linorder*

begin

definition *less-char where* [*symmetric, simp*]: *less-char = PAC-Polynomials-Term.less-char*

definition *less-eq-char where* [*symmetric, simp*]: *less-eq-char = PAC-Polynomials-Term.less-eq-char*

instance

apply *standard*

using *char.linorder-axioms*

by (*auto simp: class.linorder-def class.order-def class.preorder-def*
less-eq-char-def less-than-char-def class.order-axioms-def
class.linorder-axioms-def p2rel-def less-char-def)

end


```

instantiation list :: (linorder) linorder
begin
  definition less-list where less-list = lexordp (<)
  definition less-eq-list where less-eq-list = lexordp-eq

instance
proof standard
  have [dest]: ⟨ $\bigwedge x y :: 'a :: \text{linorder } list. (x, y) \in \text{lexord } \{(x, y). x < y\} \implies$   

             $\text{lexordp-eq } y x \implies \text{False}$ ⟩
  by (metis lexordp-antisym lexordp-conv-lexord lexordp-eq-conv-lexord)
  have [simp]: ⟨ $\bigwedge x y :: 'a :: \text{linorder } list. \text{lexordp-eq } x y \implies$   

             $\neg \text{lexordp-eq } y x \implies$   

             $(x, y) \in \text{lexord } \{(x, y). x < y\}$ ⟩
  using lexordp-conv-lexord lexordp-conv-lexordp-eq by blast
show
  ⟨ $x < y = \text{Restricted-Predicates.strict } (\leq) x y$ ⟩
  ⟨ $x \leq x$ ⟩
  ⟨ $x \leq y \implies y \leq z \implies x \leq z$ ⟩
  ⟨ $x \leq y \implies y \leq x \implies x = y$ ⟩
  ⟨ $x \leq y \vee y \leq x$ ⟩
  for x y z :: ⟨'a :: linorder list⟩
  by (auto simp: less-list-def less-eq-list-def List.lexordp-def
    lexordp-conv-lexord lexordp-into-lexordp-eq lexordp-antisym
    antisym-def lexordp-eq-refl lexordp-eq-linear intro: lexordp-eq-trans
    dest: lexordp-eq-antisym)
qed

end

```

```

lemma term-order-rel'-alt-def-lexord:
  ⟨term-order-rel' x y = ord-class.lexordp x y⟩ and
  term-order-rel'-alt-def:
  ⟨term-order-rel' x y  $\longleftrightarrow$  x < y⟩

```

```

proof –
show
  ⟨term-order-rel' x y = ord-class.lexordp x y⟩
  ⟨term-order-rel' x y  $\longleftrightarrow$  x < y⟩
  unfolding less-than-char-of-char[symmetric, abs-def]
  by (auto simp: lexordp-conv-lexord less-eq-list-def
    less-list-def lexordp-def var-order-rel-def
    rel2p-def term-order-rel-alt-def p2rel-def)
qed

```

```

lemma list-rel-list-rel-order-iff:
  assumes ⟨(a, b) ∈ ⟨string-rel⟩list-rel⟩ ⟨(a', b') ∈ ⟨string-rel⟩list-rel⟩
  shows ⟨a < a'  $\longleftrightarrow$  b < b'⟩

```

```

proof
  have H: ⟨(a, b) ∈ ⟨string-rel⟩list-rel  $\implies$   

            (a, cs) ∈ ⟨string-rel⟩list-rel  $\implies$  b = cs⟩ for cs
  using single-valued-monom-rel' IS-RIGHT-UNIQUE-string-rel
  unfolding string2-rel-def
  by (subst (asm)list-rel-sv-iff[symmetric])
       (auto simp: single-valued-def)
  assume ⟨a < a'⟩

```

then consider
 $u\ u' \text{ where } \langle a' = a @ u \# u' \rangle \mid$
 $u\ aa\ v\ w\ aaa \text{ where } \langle a = u @ aa \# v \rangle \langle a' = u @ aaa \# w \rangle \langle aa < aaa \rangle$
by (*subst (asm) less-list-def*)
(*auto simp: lexord-def List.lexordp-def*
list-rel-append1 list-rel-split-right-iff)
then show $\langle b < b' \rangle$
proof cases
case 1
then show $\langle b < b' \rangle$
using *assms*
by (*subst less-list-def*)
(*auto simp: lexord-def List.lexordp-def*
list-rel-append1 list-rel-split-right-iff dest: H)
next
case 2
then obtain $u'\ aa'\ v'\ w'\ aaa' \text{ where}$
 $\langle b = u' @ aa' \# v' \rangle \langle b' = u' @ aaa' \# w' \rangle$
 $\langle (aa, aa') \in \text{string-rel} \rangle$
 $\langle (aaa, aaa') \in \text{string-rel} \rangle$
using *assms*
by (*smt list-rel-append1 list-rel-split-right-iff single-valued-def single-valued-monom-rel*)
with $\langle aa < aaa \rangle$ **have** $\langle aa' < aaa' \rangle$
by (*auto simp: string-rel-def less-literal.rep-eq less-list-def*
lexordp-conv-lexord lexordp-def char.lexordp-conv-lexord
simp flip: less-char-def PAC-Polynomials-Term.less-char-def)
then show $\langle b < b' \rangle$
using $\langle b = u' @ aa' \# v' \rangle \langle b' = u' @ aaa' \# w' \rangle$
by (*subst less-list-def*)
(*fastforce simp: lexord-def List.lexordp-def*
list-rel-append1 list-rel-split-right-iff)
qed
next
have $H: \langle (a, b) \in \langle \text{string-rel} \rangle \text{list-rel} \implies$
 $\langle a', b \rangle \in \langle \text{string-rel} \rangle \text{list-rel} \implies a = a' \rangle$ **for** $a\ a'\ b$
using *single-valued-monom-rel'*
by (*auto simp: single-valued-def IS-LEFT-UNIQUE-def*
simp flip: inv-list-rel-eq)
assume $\langle b < b' \rangle$
then consider
 $u\ u' \text{ where } \langle b' = b @ u \# u' \rangle \mid$
 $u\ aa\ v\ w\ aaa \text{ where } \langle b = u @ aa \# v \rangle \langle b' = u @ aaa \# w \rangle \langle aa < aaa \rangle$
by (*subst (asm) less-list-def*)
(*auto simp: lexord-def List.lexordp-def*
list-rel-append1 list-rel-split-right-iff)
then show $\langle a < a' \rangle$
proof cases
case 1
then show $\langle a < a' \rangle$
using *assms*
by (*subst less-list-def*)
(*auto simp: lexord-def List.lexordp-def*
list-rel-append2 list-rel-split-left-iff dest: H)
next
case 2

```

then obtain  $u' aa' v' w' aaa'$  where
   $\langle a = u' @ aa' \# v' \rangle \langle a' = u' @ aaa' \# w' \rangle$ 
   $\langle aa', aa \rangle \in \text{string-rel}$ 
   $\langle aaa', aaa \rangle \in \text{string-rel}$ 
using assms
by (auto simp: lexord-def List.lexordp-def
  list-rel-append2 list-rel-split-left-iff dest: H)
with  $\langle aa < aaa \rangle$  have  $\langle aa' < aaa' \rangle$ 
by (auto simp: string-rel-def less-literal.rep-eq less-list-def
  lexordp-conv-lexord lexordp-def char.lexordp-conv-lexord
  simp flip: less-char-def PAC-Polynomials-Term.less-char-def)
then show  $\langle a < a' \rangle$ 
using  $\langle a = u' @ aa' \# v' \rangle \langle a' = u' @ aaa' \# w' \rangle$ 
by (subst less-list-def)
  (fastforce simp: lexord-def List.lexordp-def
  list-rel-append1 list-rel-split-right-iff)
qed
qed

```

```

lemma string-rel-le[sepref-import-param]:
  shows  $\langle ((<), (<)) \rangle \in \langle \text{string-rel} \rangle \text{list-rel} \rightarrow \langle \text{string-rel} \rangle \text{list-rel} \rightarrow \text{bool-rel}$ 
by (auto intro!: fun-relI simp: list-rel-list-rel-order-iff)

```

```

lemma [sepref-import-param]:
  assumes  $\langle \text{CONSTRAINT IS-LEFT-UNIQUE } R \rangle \langle \text{CONSTRAINT IS-RIGHT-UNIQUE } R \rangle$ 
shows  $\langle \text{remove1, remove1} \rangle \in R \rightarrow \langle R \rangle \text{list-rel} \rightarrow \langle R \rangle \text{list-rel}$ 
apply (intro fun-relI)
subgoal premises  $p$  for  $x y xs ys$ 
  using  $p(2) p(1)$  assms
by (induction xs ys rule: list-rel-induct)
  (auto simp: IS-LEFT-UNIQUE-def single-valued-def)
done

```

```

instantiation pac-step :: (heap, heap, heap) heap
begin

```

```

instance
proof standard
  obtain  $f :: \langle 'a \Rightarrow \text{nat} \rangle$  where
     $f: \langle \text{inj } f \rangle$ 
    by blast
  obtain  $g :: \langle \text{nat} \times \text{nat} \times \text{nat} \times \text{nat} \times \text{nat} \Rightarrow \text{nat} \rangle$  where
     $g: \langle \text{inj } g \rangle$ 
    by blast
  obtain  $h :: \langle 'b \Rightarrow \text{nat} \rangle$  where
     $h: \langle \text{inj } h \rangle$ 
    by blast
  obtain  $i :: \langle 'c \Rightarrow \text{nat} \rangle$  where
     $i: \langle \text{inj } i \rangle$ 
    by blast
  have [iff]:  $\langle g a = g b \longleftrightarrow a = b \rangle \langle h a'' = h b'' \longleftrightarrow a'' = b'' \rangle \langle f a' = f b' \longleftrightarrow a' = b' \rangle$ 
     $\langle i a''' = i b''' \longleftrightarrow a''' = b''' \rangle$  for  $a b a' b' a'' b'' a''' b'''$ 
  using  $f g h i$  unfolding inj-def by blast+

```

```

let ?f = ⟨λx :: ('a, 'b, 'c) pac-step.
  g (case x of
    Add a b c d ⇒ (0, i a, i b, i c, f d)
  | Del a ⇒ (1, i a, 0, 0, 0)
  | Mult a b c d ⇒ (2, i a, f b, i c, f d)
  | Extension a b c ⇒ (3, i a, f c, 0, h b))⟩
have ⟨inj ?f⟩
apply (auto simp: inj-def)
apply (case-tac x; case-tac y)
apply auto
done
then show ⟨∃f :: ('a, 'b, 'c) pac-step ⇒ nat. inj f⟩
by blast
qed

end

end
theory PAC-Assoc-Map-Rel
imports PAC-Map-Rel
begin

```

11 Hash Map as association list

```

type-synonym ('k, 'v) hash-assoc = ⟨('k × 'v) list⟩

```

```

definition hassoc-map-rel-raw :: ⟨(('k, 'v) hash-assoc × -) set⟩ where
  ⟨hassoc-map-rel-raw = br map-of (λ-. True)⟩

```

```

abbreviation hassoc-map-assn :: ⟨('k ⇒ 'v option) ⇒ ('k, 'v) hash-assoc ⇒ assn⟩ where
  ⟨hassoc-map-assn ≡ pure (hassoc-map-rel-raw)⟩

```

```

lemma hassoc-map-rel-raw-empty[simp]:
  ⟨([], m) ∈ hassoc-map-rel-raw ⟷ m = Map.empty⟩
  ⟨(p, Map.empty) ∈ hassoc-map-rel-raw ⟷ p = []⟩
  ⟨hassoc-map-assn Map.empty [] = emp⟩
by (auto simp: hassoc-map-rel-raw-def br-def pure-def)

```

```

definition hassoc-new :: ⟨('k, 'v) hash-assoc Heap⟩ where
  ⟨hassoc-new = return []⟩

```

```

lemma precise-hassoc-map-assn: ⟨precise hassoc-map-assn⟩
by (auto intro!: precise-pure)
  (auto simp: single-valued-def hassoc-map-rel-raw-def
  br-def)

```

```

definition hassoc-isEmpty :: ⟨('k × 'v) list ⇒ bool Heap⟩ where
  hassoc-isEmpty ht ≡ return (length ht = 0)

```

```

interpretation hassoc: bind-map-empty hassoc-map-assn hassoc-new
by unfold-locales
  (auto intro: precise-hassoc-map-assn
  simp: ent-refl-true hassoc-new-def
  intro!: return-cons-rule)

```

interpretation *hassoc: bind-map-is-empty hassoc-map-assn hassoc-isEmpty*
by *unfold-locales*
(auto simp: precise-hassoc-map-assn hassoc-isEmpty-def ent-refl-true
intro!: precise-pure return-cons-rule)

definition *op-assoc-empty* \equiv *IICF-Map.op-map-empty*

interpretation *hassoc: map-custom-empty op-assoc-empty*
by *unfold-locales (simp add: op-assoc-empty-def)*

lemmas *[sepref-fr-rules] = hassoc.empty-hnr[folded op-assoc-empty-def]*

definition *hassoc-update* $:: 'k \Rightarrow 'v \Rightarrow ('k, 'v) \text{ hash-assoc} \Rightarrow ('k, 'v) \text{ hash-assoc Heap}$ **where**
hassoc-update k v ht = return ((k, v) # ht)

lemma *hassoc-map-assn-Cons*:
 $\langle \text{hassoc-map-assn } (m) (p) \Rightarrow_A \text{hassoc-map-assn } (m(k \mapsto v)) ((k, v) \# p) * \text{true} \rangle$
by *(auto simp: hassoc-map-rel-raw-def pure-def br-def)*

interpretation *hassoc: bind-map-update hassoc-map-assn hassoc-update*
by *unfold-locales*
(auto intro!: return-cons-rule
simp: hassoc-update-def hassoc-map-assn-Cons)

definition *hassoc-delete* $:: \langle 'k \Rightarrow ('k, 'v) \text{ hash-assoc} \Rightarrow ('k, 'v) \text{ hash-assoc Heap} \rangle$ **where**
 $\langle \text{hassoc-delete } k \text{ ht} = \text{return } (\text{filter } (\lambda(a, b). a \neq k) \text{ ht}) \rangle$

lemma *hassoc-map-of-filter-all*:
 $\langle \text{map-of } p \mid' (- \{k\}) = \text{map-of } (\text{filter } (\lambda(a, b). a \neq k) p) \rangle$
apply *(induction p)*
apply *(auto simp: restrict-map-def fun-eq-iff split: if-split)*
apply *presburger+*
done

lemma *hassoc-map-assn-hassoc-delete*: $\langle \langle \text{hassoc-map-assn } m \ p \rangle \text{hassoc-delete } k \ p \ \langle \text{hassoc-map-assn } (m \mid' (- \{k\})) \rangle_t \rangle$
by *(auto simp: hassoc-delete-def hassoc-map-rel-raw-def pure-def br-def*
hassoc-map-of-filter-all
intro!: return-cons-rule)

interpretation *hassoc: bind-map-delete hassoc-map-assn hassoc-delete*
by *unfold-locales*
(auto intro: hassoc-map-assn-hassoc-delete)

definition *hassoc-lookup* $:: \langle 'k \Rightarrow ('k, 'v) \text{ hash-assoc} \Rightarrow 'v \text{ option Heap} \rangle$ **where**
 $\langle \text{hassoc-lookup } k \text{ ht} = \text{return } (\text{map-of } \text{ht } k) \rangle$

lemma *hassoc-map-assn-hassoc-lookup*:
 $\langle \langle \text{hassoc-map-assn } m \ p \rangle \text{hassoc-lookup } k \ p \ \langle \lambda r. \text{hassoc-map-assn } m \ p * \uparrow (r = m \ k) \rangle_t \rangle$
by *(auto simp: hassoc-lookup-def hassoc-map-rel-raw-def pure-def br-def)*

hassoc-map-of-filter-all
intro!: return-cons-rule)

interpretation *hassoc: bind-map-lookup hassoc-map-assn hassoc-lookup*
by *unfold-locales*
(rule hassoc-map-assn-hassoc-lookup)

setup *Locale-Code.open-block*
interpretation *hassoc: gen-contains-key-by-lookup hassoc-map-assn hassoc-lookup*
by *unfold-locales*
setup *Locale-Code.close-block*

interpretation *hassoc: bind-map-contains-key hassoc-map-assn hassoc.contains-key*
by *unfold-locales*

11.1 Conversion from assoc to other map

definition *hash-of-assoc-map where*
 $\langle \text{hash-of-assoc-map } xs = \text{fold } (\lambda(k, v) m. \text{if } m \ k \neq \text{None then } m \text{ else } m(k \mapsto v)) \ xs \ \text{Map.empty} \rangle$

lemma *map-upd-map-add-left:*
 $\langle m(a \mapsto b) ++ m' = m ++ (\text{if } a \notin \text{dom } m' \text{ then } m'(a \mapsto b) \text{ else } m') \rangle$

proof –
have $\langle m' a = \text{Some } y \implies m(a \mapsto b) ++ m' = m ++ m' \rangle$ **for** *y*
by (*metis (no-types) fun-upd-triv fun-upd-upd map-add-assoc map-add-empty map-add-upd map-le-iff-map-add-commute*)
then show *?thesis*
by *auto*
qed

lemma *fold-map-of-alt:*
 $\langle \text{fold } (\lambda(k, v) m. \text{if } m \ k \neq \text{None then } m \text{ else } m(k \mapsto v)) \ xs \ m' = \text{map-of } xs ++ m' \rangle$
by (*induction xs arbitrary: m'*)
(auto simp: map-upd-map-add-left)

lemma *map-of-alt-def:*
 $\langle \text{map-of } xs = \text{hash-of-assoc-map } xs \rangle$
using *fold-map-of-alt[of xs Map.empty]*
unfolding *hash-of-assoc-map-def*
by *auto*

definition *hashmap-conv where*
[simp]: $\langle \text{hashmap-conv } x = x \rangle$

lemma *hash-of-assoc-map-id:*
 $\langle (\text{hash-of-assoc-map}, \text{hashmap-conv}) \in \text{hassoc-map-rel-raw} \rightarrow \text{Id} \rangle$
by (*auto intro!: fun-relI simp: hassoc-map-rel-raw-def br-def map-of-alt-def*)

definition *hassoc-map-rel where*
hassoc-map-rel-internal-def:
 $\langle \text{hassoc-map-rel } K \ V = \text{hassoc-map-rel-raw } O \ \langle K, V \rangle \text{map-rel} \rangle$

lemma *hassoc-map-rel-def:*
 $\langle \langle K, V \rangle \text{hassoc-map-rel} = \text{hassoc-map-rel-raw } O \ \langle K, V \rangle \text{map-rel} \rangle$
unfolding *relAPP-def hassoc-map-rel-internal-def*
by *auto*

end

```
theory PAC-Checker-Init
  imports PAC-Checker WB-Sort PAC-Checker-Relation
begin
```

12 Initial Normalisation of Polynomials

12.1 Sorting

Adapted from the theory *HOL-ex.MergeSort* by Tobias Nipkow. We did not change much, but we refine it to executable code and try to improve efficiency.

```
fun merge :: 'a list ⇒ 'a list ⇒ 'a list
where
  merge f (x#xs) (y#ys) =
    (if f x y then x # merge f xs (y#ys) else y # merge f (x#xs) ys)
| merge f xs [] = xs
| merge f [] ys = ys
```

```
lemma mset-merge [simp]:
  mset (merge f xs ys) = mset xs + mset ys
by (induct f xs ys rule: merge.induct) (simp-all add: ac-simps)
```

```
lemma set-merge [simp]:
  set (merge f xs ys) = set xs ∪ set ys
by (induct f xs ys rule: merge.induct) auto
```

```
lemma sorted-merge:
  transp f ⇒ (∧ x y. f x y ∨ f y x) ⇒
  sorted-wrt f (merge f xs ys) ↔ sorted-wrt f xs ∧ sorted-wrt f ys
apply (induct f xs ys rule: merge.induct)
apply (auto simp add: ball-Un not-le less-le dest: transpD)
apply blast
apply (blast dest: transpD)
done
```

```
fun msort :: 'a list ⇒ 'a list
where
  msort f [] = []
| msort f [x] = [x]
| msort f xs = merge f
  (msort f (take (size xs div 2) xs))
  (msort f (drop (size xs div 2) xs))
```

```
fun swap-ternary :: 'a × 'a × 'a ⇒ 'a × 'a × 'a where
  ⟨swap-ternary f m n =
    (if (m = 0 ∧ n = 1)
      then (λ(a, b, c). if f a b then (a, b, c)
        else (b,a,c))
      else if (m = 0 ∧ n = 2)
        then (λ(a, b, c). if f a c then (a, b, c)
          else (c,b,a))
```

```

else if (m = 1 ∧ n = 2)
then (λ(a, b, c). if f b c then (a, b, c)
      else (a,c,b))
else (λ(a, b, c). (a,b,c)))

```

fun *msort2* :: - ⇒ 'a list ⇒ 'a list

where

```

msort2 f [] = []
| msort2 f [x] = [x]
| msort2 f [x,y] = (if f x y then [x,y] else [y,x])
| msort2 f xs = merge f
                  (msort f (take (size xs div 2) xs))
                  (msort f (drop (size xs div 2) xs))

```

lemmas [*code del*] =

msort2.simps

declare *msort2.simps*[*simp del*]

lemmas [*code*] =

msort2.simps[*unfolded swap-ternary.simps, simplified*]

declare *msort2.simps*[*simp*]

lemma *msort-msort2*:

fixes *xs* :: ⟨'a :: linorder list⟩

shows ⟨*msort* (≤) *xs* = *msort2* (≤) *xs*⟩

apply (*induction* ⟨(≤) :: 'a ⇒ 'a ⇒ bool⟩ *xs* rule: *msort2.induct*)

apply (*auto dest: transpD*)

done

lemma *sorted-msort*:

transp f ⇒ (∧*x y*. *f x y* ∨ *f y x*) ⇒

sorted-wrt f (*msort f xs*)

by (*induct f xs* rule: *msort.induct*) (*simp-all add: sorted-merge*)

lemma *mset-msort*[*simp*]:

mset (*msort f xs*) = *mset xs*

by (*induction f xs* rule: *msort.induct*)

(*simp-all add: union-code*)

12.2 Sorting applied to monomials

lemma *merge-coeffs-alt-def*:

⟨(*RETURN* o *merge-coeffs*) *p* =

RECT(λ*f p*.

(*case p* of

[] ⇒ *RETURN* []

| [-] => *RETURN p*

| ((*xs*, *n*) # (*ys*, *m*) # *p*) ⇒

(*if xs* = *ys*

then *if n* + *m* ≠ 0 then *f* ((*xs*, *n* + *m*) # *p*) else *f p*

else do {*p* ← *f* ((*ys*, *m*) # *p*); *RETURN* ((*xs*, *n*) # *p*)})

p⟩

apply (*induction p* rule: *merge-coeffs.induct*)

subgoal **by** (*subst RECT-unfold, refine-mono*) *auto*

subgoal **by** (*subst RECT-unfold, refine-mono*) *auto*

subgoal for $x p y q$
by (*subst RECT-unfold, refine-mono*)
(smt case-prod-conv list.simps(5) merge-coeffs.simps(3) nres-monad1
push-in-let-conv(2))
done

lemma *hn-invalid-recover*:
 $\langle is-pure R \implies hn-invalid R = (\lambda x y. R x y * true) \rangle$
 $\langle is-pure R \implies invalid-assn R = (\lambda x y. R x y * true) \rangle$
by (*auto simp: is-pure-conv invalid-pure-recover hn-ctxt-def intro!: ext*)

lemma *safe-poly-vars*:
shows
[safe-constraint-rules]:
is-pure (poly-assn) and
[safe-constraint-rules]:
is-pure (monom-assn) and
[safe-constraint-rules]:
is-pure (monomial-assn) and
[safe-constraint-rules]:
is-pure string-assn
by (*auto intro!: pure-prod list-assn-pure simp: prod-assn-pure-conv*)

lemma *invalid-assn-distrib*:
 $\langle invalid-assn monom-assn \times_a invalid-assn int-assn = invalid-assn (monom-assn \times_a int-assn) \rangle$
apply (*simp add: invalid-pure-recover hn-invalid-recover*
safe-constraint-rules)
apply (*subst hn-invalid-recover*)
apply (*rule safe-poly-vars(2)*)
apply (*subst hn-invalid-recover*)
apply (*rule safe-poly-vars*)
apply (*auto intro!: ext*)
done

lemma *WTF-RF-recover*:
 $\langle hn-ctxt (invalid-assn monom-assn \times_a invalid-assn int-assn) xb$
 $x'a \vee_A$
 $hn-ctxt monomial-assn xb x'a \implies_t$
 $hn-ctxt (monomial-assn) xb x'a \rangle$
by (*smt assn-aci(5) hn-ctxt-def invalid-assn-distrib invalid-pure-recover is-pure-conv*
merge-thms(4) merge-true-star reorder-enttI safe-poly-vars(3) star-aci(2) star-aci(3))

lemma *WTF-RF*:
 $\langle hn-ctxt (invalid-assn monom-assn \times_a invalid-assn int-assn) xb x'a *$
 $(hn-invalid poly-assn la l'a * hn-invalid int-assn a2' a2 *$
 $hn-invalid monom-assn a1' a1 *$
 $hn-invalid poly-assn l l' *$
 $hn-invalid monomial-assn xa x' *$
 $hn-invalid poly-assn ax px) \implies_t$
 $hn-ctxt (monomial-assn) xb x'a *$
 $hn-ctxt poly-assn$
 $la l'a *$
 $hn-ctxt poly-assn l l' *$
 $(hn-invalid int-assn a2' a2 *$
 $hn-invalid monom-assn a1' a1 *$

```

    hn-invalid monomial-assn xa x' *
    hn-invalid poly-assn ax px)
  ⟨hn-ctxt (invalid-assn monom-assn ×a invalid-assn int-assn) xa x' *
    (hn-ctxt poly-assn l l' * hn-invalid poly-assn ax px) ⇒t
    hn-ctxt (monomial-assn) xa x' *
    hn-ctxt poly-assn l l' *
    hn-ctxt poly-assn ax px *
    emp⟩
  by sepref-dbg-trans-step+

```

The refinement framework is completely lost here when synthesizing the constants – it does not understand what is pure (actually everything) and what must be destroyed.

```

sepref-definition merge-coeffs-impl
  is ⟨RETURN o merge-coeffs⟩
  :: ⟨poly-assnd →a poly-assn⟩
  supply [[goals-limit=1]]
  unfolding merge-coeffs-alt-def
    HOL-list.fold-custom-empty poly-assn-alt-def
  apply (rewrite in ⟨-⟩ annotate-assn[where A=⟨poly-assn⟩])
  apply sepref-dbg-preproc
  apply sepref-dbg-cons-init
  apply sepref-dbg-id
  apply sepref-dbg-monadify
  apply sepref-dbg-opt-init
  apply (rule WTF-RF | sepref-dbg-trans-step)+
  apply sepref-dbg-opt
  apply sepref-dbg-cons-solve
  apply sepref-dbg-cons-solve
  apply sepref-dbg-constraints
  done

```

```

definition full-quicksort-poly where
  ⟨full-quicksort-poly = full-quicksort-ref (λx y. x = y ∨ (x, y) ∈ term-order-rel) fst⟩

```

```

lemma down-eq-id-list-rel: ⟨↓((Id)list-rel) x = x⟩
  by auto

```

```

definition quicksort-poly:: ⟨nat ⇒ nat ⇒ llist-polynomial ⇒ (llist-polynomial) nres⟩ where
  ⟨quicksort-poly x y z = quicksort-ref (≤) fst (x, y, z)⟩

```

```

term partition-between-ref

```

```

definition partition-between-poly :: ⟨nat ⇒ nat ⇒ llist-polynomial ⇒ (llist-polynomial × nat) nres⟩
where
  ⟨partition-between-poly = partition-between-ref (≤) fst⟩

```

```

definition partition-main-poly :: ⟨nat ⇒ nat ⇒ llist-polynomial ⇒ (llist-polynomial × nat) nres⟩ where
  ⟨partition-main-poly = partition-main (≤) fst⟩

```

```

lemma string-list-trans:
  ⟨(xa :: char list list, ya) ∈ lexord (lexord {(x, y). x < y}) ⇒
  (ya, z) ∈ lexord (lexord {(x, y). x < y}) ⇒
  (xa, z) ∈ lexord (lexord {(x, y). x < y})⟩
  by (smt less-char-def char.less-trans less-than-char-def lexord-partial-trans p2rel-def)

```

```

lemma full-quicksort-sort-poly-spec:
  ⟨(full-quicksort-poly, sort-poly-spec) ∈ ⟨Id⟩list-rel →f ⟨⟨Id⟩list-rel⟩nres-rel⟩
proof –
  have xs: ⟨(xs, xs) ∈ ⟨Id⟩list-rel⟩ and ⟨↓(⟨Id⟩list-rel) x = x⟩ for x xs
    by auto
  show ?thesis
    apply (intro frefI nres-relI)
    unfolding full-quicksort-poly-def
    apply (rule full-quicksort-ref-full-quicksort[THEN fref-to-Down-curry, THEN order-trans])
    subgoal
      by (auto simp: rel2p-def var-order-rel-def p2rel-def Relation.total-on-def
        dest: string-list-trans)
    subgoal
      using total-on-lexord-less-than-char-linear[unfolded var-order-rel-def]
      apply (auto simp: rel2p-def var-order-rel-def p2rel-def Relation.total-on-def less-char-def)
      done
    subgoal by fast
    apply (rule xs)
    apply (subst down-eq-id-list-rel)
    unfolding sorted-wrt-map sort-poly-spec-def
    apply (rule full-quicksort-correct-sorted[where R = ⟨(λx y. x = y ∨ (x, y) ∈ term-order-rel)⟩] and
      h = ⟨fst⟩,
        THEN order-trans])
    subgoal
      by (auto simp: rel2p-def var-order-rel-def p2rel-def Relation.total-on-def dest: string-list-trans)
    subgoal for x y
      using total-on-lexord-less-than-char-linear[unfolded var-order-rel-def]
      apply (auto simp: rel2p-def var-order-rel-def p2rel-def Relation.total-on-def
        less-char-def)
      done
    subgoal
      by (auto simp: rel2p-def p2rel-def)
    done
qed

```

12.3 Lifting to polynomials

definition *merge-sort-poly* :: ⟨-⟩ **where**
 ⟨*merge-sort-poly* = *msort* (λa b. *fst* a ≤ *fst* b)⟩

definition *merge-monoms-poly* :: ⟨-⟩ **where**
 ⟨*merge-monoms-poly* = *msort* (≤)⟩

definition *merge-poly* :: ⟨-⟩ **where**
 ⟨*merge-poly* = *merge* (λa b. *fst* a ≤ *fst* b)⟩

definition *merge-monoms* :: ⟨-⟩ **where**
 ⟨*merge-monoms* = *merge* (≤)⟩

definition *msort-poly-impl* :: ⟨(String.literal list × int) list ⇒ -⟩ **where**
 ⟨*msort-poly-impl* = *msort* (λa b. *fst* a ≤ *fst* b)⟩

definition *msort-monoms-impl* :: ⟨(String.literal list) ⇒ -⟩ **where**
 ⟨*msort-monoms-impl* = *msort* (≤)⟩

lemma *msort-poly-impl-alt-def*:

```

⟨msort-poly-impl xs =
  (case xs of
    [] ⇒ []
  | [a] ⇒ [a]
  | [a,b] ⇒ if fst a ≤ fst b then [a,b] else [b,a]
  | xs ⇒ merge-poly
    (msort-poly-impl (take ((length xs) div 2) xs))
    (msort-poly-impl (drop ((length xs) div 2) xs)))⟩
unfolding msort-poly-impl-def
apply (auto split: list.splits simp: merge-poly-def)
done

```

```

lemma le-term-order-rel':
  ⟨(≤) = (λx y. x = y ∨ term-order-rel' x y)⟩
apply (intro ext)
apply (auto simp add: less-list-def less-eq-list-def
  lexordp-eq-conv-lexord lexordp-def)
using term-order-rel'-alt-def-lexord term-order-rel'-def apply blast
using term-order-rel'-alt-def-lexord term-order-rel'-def apply blast
done

```

```

fun lexord-eq where
  ⟨lexord-eq [] - = True⟩ |
  ⟨lexord-eq (x # xs) (y # ys) = (x < y ∨ (x = y ∧ lexord-eq xs ys))⟩ |
  ⟨lexord-eq - - = False⟩

```

```

lemma [simp]:
  ⟨lexord-eq [] [] = True⟩
  ⟨lexord-eq (a # b) [] = False⟩
  ⟨lexord-eq [] (a # b) = True⟩
apply auto
done

```

```

lemma var-order-rel':
  ⟨(≤) = (λx y. x = y ∨ (x,y) ∈ var-order-rel)⟩
by (intro ext)
  (auto simp add: less-list-def less-eq-list-def
  lexordp-eq-conv-lexord lexordp-def var-order-rel-def
  lexordp-conv-lexord p2rel-def)

```

```

lemma var-order-rel'':
  ⟨(x,y) ∈ var-order-rel ⟷ x < y⟩
by (metis leD less-than-char-linear lexord-linear neq-iff var-order-rel' var-order-rel-antisym
  var-order-rel-def)

```

```

lemma lexord-eq-alt-def1:
  ⟨a ≤ b = lexord-eq a b⟩ for a b :: ⟨String.literal list⟩
unfolding le-term-order-rel'
apply (induction a b rule: lexord-eq.induct)
apply (auto simp: var-order-rel'' less-eq-list-def)
done

```

```

lemma lexord-eq-alt-def2:
  ⟨(RETURN oo lexord-eq) xs ys =

```

```

RECT (λf (xs, ys).
  case (xs, ys) of
    ([], -) ⇒ RETURN True
  | (x # xs, y # ys) ⇒
    if x < y then RETURN True
    else if x = y then f (xs, ys) else RETURN False
  | - ⇒ RETURN False)
(xs, ys)
apply (subst eq-commute)
apply (induction xs ys rule: lexord-eq.induct)
subgoal by (subst RECT-unfold, refine-mono) auto
subgoal by (subst RECT-unfold, refine-mono) auto
subgoal by (subst RECT-unfold, refine-mono) auto
done

```

definition *var-order'* **where**

```
[simp]: ⟨var-order' = var-order⟩
```

lemma *var-order-rel*[*def-pat-rules*]:

```
⟨(∈)$x,y)$var-order-rel ≡ var-order'$x$y⟩
by (auto simp: p2rel-def rel2p-def)
```

lemma *var-order-rel-alt-def*:

```
⟨var-order-rel = p2rel char.lexordp⟩
apply (auto simp: p2rel-def char.lexordp-conv-lexord var-order-rel-def)
using char.lexordp-conv-lexord apply auto
done
```

lemma *var-order-rel-var-order*:

```
⟨(x, y) ∈ var-order-rel ⟷ var-order x y⟩
by (auto simp: rel2p-def)
```

lemma *var-order-string-le*[*sepref-import-param*]:

```
⟨((<), var-order') ∈ string-rel → string-rel → bool-rel⟩
apply (auto intro!: freI simp: string-rel-def String.less-literal-def
  rel2p-def linorder.lexordp-conv-lexord[OF char.linorder-axioms,
  unfolded less-eq-char-def] var-order-rel-def
  p2rel-def
  simp flip: PAC-Polynomials-Term.less-char-def)
using char.lexordp-conv-lexord apply auto
done
```

lemma [*sepref-import-param*]:

```
⟨(≤), (≤)⟩ ∈ monom-rel → monom-rel → bool-rel⟩
apply (intro fun-relI)
using list-rel-list-rel-order-iff by fastforce
```

lemma [*sepref-import-param*]:

```
⟨(<), (<)⟩ ∈ string-rel → string-rel → bool-rel⟩
```

proof –

```
have [iff]: ⟨ord.lexordp (<) (literal.explode a) (literal.explode aa) ⟷
  List.lexordp (<) (literal.explode a) (literal.explode aa)⟩ for a aa
apply (rule iffI)
apply (metis PAC-Checker-Relation.less-char-def char.lexordp-conv-lexord less-list-def)
```

```

    p2rel-def var-order-rel'' var-order-rel-def)
apply (metis PAC-Checker-Relation.less-char-def char.lexordp-conv-lexord less-list-def
    p2rel-def var-order-rel'' var-order-rel-def)
done
show ?thesis
unfolding string-rel-def less-literal.rep-eq less-than-char-def
    less-eq-list-def PAC-Polynomials-Term.less-char-def[symmetric]
by (intro fun-relI)
    (auto simp: string-rel-def less-literal.rep-eq
    less-list-def char.lexordp-conv-lexord lexordp-eq-refl
    lexordp-eq-conv-lexord)
qed

```

```

lemma lexordp-char-char: ⟨ord-class.lexordp = char.lexordp⟩
unfolding char.lexordp-def ord-class.lexordp-def
by (rule arg-cong[of - - lfp])
    (auto intro!: ext)

```

```

lemma [sepref-import-param]:
    ⟨(≤), (≤)⟩ ∈ string-rel → string-rel → bool-rel
unfolding string-rel-def less-eq-literal.rep-eq less-than-char-def
    less-eq-list-def PAC-Polynomials-Term.less-char-def[symmetric]
by (intro fun-relI)
    (auto simp: string-rel-def less-eq-literal.rep-eq less-than-char-def
    less-eq-list-def char.lexordp-eq-conv-lexord lexordp-eq-refl
    lexordp-eq-conv-lexord lexordp-char-char
    simp flip: less-char-def[abs-def])

```

```

sepref-register lexord-eq
sepref-definition lexord-eq-term
is ⟨uncurry (RETURN oo lexord-eq)⟩
    :: ⟨monom-assnk *a monom-assnk →a bool-assn⟩
supply[[goals-limit=1]]
unfolding lexord-eq-alt-def2
by sepref

```

```

declare lexord-eq-term.refine[sepref-fr-rules]

```

```

lemmas [code del] = msort-poly-impl-def msort-monom-impl-def
lemmas [code] =
    msort-poly-impl-def[unfolded lexord-eq-alt-def1[abs-def]]
    msort-monom-impl-def[unfolded msort-msort2]

```

```

lemma term-order-rel-trans:
    ⟨(a, aa) ∈ term-order-rel ⇒
    (aa, ab) ∈ term-order-rel ⇒ (a, ab) ∈ term-order-rel⟩
by (metis PAC-Checker-Relation.less-char-def p2rel-def string-list-trans var-order-rel-def)

```

```

lemma merge-sort-poly-sort-poly-spec:
    ⟨(RETURN o merge-sort-poly, sort-poly-spec) ∈ ⟨Id⟩list-rel →f ⟨⟨Id⟩list-rel⟩nres-rel⟩
unfolding sort-poly-spec-def merge-sort-poly-def
apply (intro frefI nres-relI)
using total-on-lexord-less-than-char-linear var-order-rel-def

```

by (*auto intro!*: *sorted-msort simp*: *sorted-wrt-map rel2p-def*
le-term-order-rel' *transp-def* *dest*: *term-order-rel-trans*)

lemma *msort-alt-def*:

⟨*RETURN* *o* (*msort* *f*) =
RECT (λg *xs*.
case *xs* *of*
 [] \Rightarrow *RETURN* []
 | [*x*] \Rightarrow *RETURN* [*x*]
 | - \Rightarrow *do* {
 a \leftarrow *g* (*take* (*size* *xs* *div* 2) *xs*);
 b \leftarrow *g* (*drop* (*size* *xs* *div* 2) *xs*);
 RETURN (*merge* *f* *a* *b*)}⟩

apply (*intro ext*)

unfolding *comp-def*

apply (*induct-tac* *f* *x* *rule*: *msort.induct*)

subgoal **by** (*subst* *RECT-unfold*, *refine-mono*) *auto*

subgoal **by** (*subst* *RECT-unfold*, *refine-mono*) *auto*

subgoal

by (*subst* *RECT-unfold*, *refine-mono*)

(*smt let-to-bind-conv* *list.simps*(5) *msort.simps*(3))

done

lemma *monomial-rel-order-map*:

⟨(*x*, *a*, *b*) \in *monomial-rel* \implies
 (*y*, *aa*, *bb*) \in *monomial-rel* \implies
fst *x* \leq *fst* *y* \iff *a* \leq *aa*⟩

apply (*cases* *x*; *cases* *y*)

apply *auto*

using *list-rel-list-rel-order-iff* **by** *fastforce+*

lemma *step-rewrite-pure*:

fixes *K* :: ⟨('olbl \times 'lbl) *set*⟩

shows

⟨*pure* (*p2rel* ((*K*, *V*, *R*)*pac-step-rel-raw*)) = *pac-step-rel-assn* (*pure* *K*) (*pure* *V*) (*pure* *R*)⟩
 ⟨*monomial-assn* = *pure* (*monom-rel* \times_r *int-rel*)⟩ **and**

poly-assn-list:

⟨*poly-assn* = *pure* ((*monom-rel* \times_r *int-rel*)*list-rel*)⟩

subgoal

apply (*intro ext*)

apply (*case-tac* *x*; *case-tac* *xa*)

apply (*auto simp*: *relAPP-def* *p2rel-def* *pure-def*)

done

subgoal *H*

apply (*intro ext*)

apply (*case-tac* *x*; *case-tac* *xa*)

by (*simp add*: *list-assn-pure-conv*)

subgoal

unfolding *H*

by (*simp add*: *list-assn-pure-conv* *relAPP-def*)

done

lemma *safe-pac-step-rel-assn[safe-constraint-rules]*:

is-pure *K* \implies *is-pure* *V* \implies *is-pure* *R* \implies *is-pure* (*pac-step-rel-assn* *K* *V* *R*)

by (auto simp: step-rewrite-pure(1)[symmetric] is-pure-conv)

lemma merge-poly-merge-poly:

⟨(merge-poly, merge-poly)
∈ poly-rel → poly-rel → poly-rel⟩
unfolding merge-poly-def
apply (intro fun-relI)
subgoal for a a' aa a'a
apply (induction ⟨(λ(a :: String.literal list × int)
(b :: String.literal list × int). fst a ≤ fst b)⟩ a aa
arbitrary: a' a'a
rule: merge.induct)
subgoal
by (auto elim!: list-relE3 list-relE4 list-relE list-relE2
simp: monomial-rel-order-map)
subgoal
by (auto elim!: list-relE3 list-relE)
subgoal
by (auto elim!: list-relE3 list-relE4 list-relE list-relE2)
done
done

lemmas [fcomp-norm-unfold] =
poly-assn-list[symmetric]
step-rewrite-pure(1)

lemma merge-poly-merge-poly2:

⟨(a, b) ∈ poly-rel ⟹ (a', b') ∈ poly-rel ⟹
(merge-poly a a', merge-poly b b') ∈ poly-rel⟩
using merge-poly-merge-poly
unfolding fun-rel-def
by auto

lemma list-rel-takeD:

⟨(a, b) ∈ ⟨R⟩list-rel ⟹ (n, n') ∈ Id ⟹ (take n a, take n' b) ∈ ⟨R⟩list-rel⟩
by (simp add: list-rel-eq-listrel listrel-iff-nth relAPP-def)

lemma list-rel-dropD:

⟨(a, b) ∈ ⟨R⟩list-rel ⟹ (n, n') ∈ Id ⟹ (drop n a, drop n' b) ∈ ⟨R⟩list-rel⟩
by (simp add: list-rel-eq-listrel listrel-iff-nth relAPP-def)

lemma merge-sort-poly[seprel-import-param]:

⟨(msort-poly-impl, merge-sort-poly)
∈ poly-rel → poly-rel⟩
unfolding merge-sort-poly-def msort-poly-impl-def
apply (intro fun-relI)
subgoal for a a'
apply (induction ⟨(λ(a :: String.literal list × int)
(b :: String.literal list × int). fst a ≤ fst b)⟩ a
arbitrary: a'
rule: msort.induct)
subgoal
by auto
subgoal


```

    by (auto elim!: list-relE3 list-relE)
  subgoal premises p
    using p
    by (auto elim!: list-relE3 list-relE4 list-relE list-relE2
        simp: merge-poly-def[symmetric]
        intro!: list-rel-takeD list-rel-dropD
        intro!: merge-poly-merge-poly2 p(1)[simplified] p(2)[simplified],
        auto simp: list-rel-imp-same-length)
  done
done

```

lemmas [sepref-fr-rules] = merge-sort-poly[FCOMP merge-sort-poly-sort-poly-spec]

```

sepref-definition partition-main-poly-impl
  is ⟨uncurry2 partition-main-poly⟩
  :: ⟨nat-assnk *a nat-assnk *a poly-assnk →a prod-assn poly-assn nat-assn ⟩
  unfolding partition-main-poly-def partition-main-def
    term-order-rel'-def[symmetric]
    term-order-rel'-alt-def
    le-term-order-rel'
  by sepref

```

declare partition-main-poly-impl.refine[sepref-fr-rules]

```

sepref-definition partition-between-poly-impl
  is ⟨uncurry2 partition-between-poly⟩
  :: ⟨nat-assnk *a nat-assnk *a poly-assnk →a prod-assn poly-assn nat-assn ⟩
  unfolding partition-between-poly-def partition-between-ref-def
    partition-main-poly-def[symmetric]
  unfolding choose-pivot3-def
    term-order-rel'-def[symmetric]
    term-order-rel'-alt-def choose-pivot-def
    lexord-eq-alt-def1
  by sepref

```

declare partition-between-poly-impl.refine[sepref-fr-rules]

```

sepref-definition quicksort-poly-impl
  is ⟨uncurry2 quicksort-poly⟩
  :: ⟨nat-assnk *a nat-assnk *a poly-assnk →a poly-assn ⟩
  unfolding partition-main-poly-def quicksort-ref-def quicksort-poly-def
    partition-between-poly-def[symmetric]
  by sepref

```

lemmas [sepref-fr-rules] = quicksort-poly-impl.refine

```

sepref-register quicksort-poly
sepref-definition full-quicksort-poly-impl
  is ⟨full-quicksort-poly⟩
  :: ⟨poly-assnk →a poly-assn ⟩
  unfolding full-quicksort-poly-def full-quicksort-ref-def
    quicksort-poly-def[symmetric]
    le-term-order-rel'[symmetric]

```

$term\text{-}order\text{-}rel'\text{-}def[symmetric]$
 $List.null\text{-}def$
by $sepref$

lemmas $sort\text{-}poly\text{-}spec\text{-}hnr =$
 $full\text{-}quicksort\text{-}poly\text{-}impl.refine[FCOMP full\text{-}quicksort\text{-}sort\text{-}poly\text{-}spec]$

declare $merge\text{-}coeffs\text{-}impl.refine[sepref\text{-}fr\text{-}rules]$

sepref-definition $normalize\text{-}poly\text{-}impl$
is $\langle normalize\text{-}poly \rangle$
 $:: \langle poly\text{-}assn^k \rightarrow_a poly\text{-}assn \rangle$
supply $[[goals\text{-}limit=1]]$
unfolding $normalize\text{-}poly\text{-}def$
by $sepref$

declare $normalize\text{-}poly\text{-}impl.refine[sepref\text{-}fr\text{-}rules]$

definition $full\text{-}quicksort\text{-}vars$ **where**
 $\langle full\text{-}quicksort\text{-}vars = full\text{-}quicksort\text{-}ref (\lambda x y. x = y \vee (x, y) \in var\text{-}order\text{-}rel) id \rangle$

definition $quicksort\text{-}vars:: \langle nat \Rightarrow nat \Rightarrow string\ list \Rightarrow (string\ list) nres \rangle$ **where**
 $\langle quicksort\text{-}vars\ x\ y\ z = quicksort\text{-}ref (\leq) id\ (x, y, z) \rangle$

definition $partition\text{-}between\text{-}vars :: \langle nat \Rightarrow nat \Rightarrow string\ list \Rightarrow (string\ list \times nat) nres \rangle$ **where**
 $\langle partition\text{-}between\text{-}vars = partition\text{-}between\text{-}ref (\leq) id \rangle$

definition $partition\text{-}main\text{-}vars :: \langle nat \Rightarrow nat \Rightarrow string\ list \Rightarrow (string\ list \times nat) nres \rangle$ **where**
 $\langle partition\text{-}main\text{-}vars = partition\text{-}main (\leq) id \rangle$

lemma $total\text{-}on\text{-}lexord\text{-}less\text{-}than\text{-}char\text{-}linear2:$
 $\langle xs \neq ys \implies (xs, ys) \notin lexord\ (less\text{-}than\text{-}char) \longleftrightarrow$
 $(ys, xs) \in lexord\ less\text{-}than\text{-}char \rangle$
using $lexord\text{-}linear[of\ \langle less\text{-}than\text{-}char \rangle xs\ ys]$
using $lexord\text{-}linear[of\ \langle less\text{-}than\text{-}char \rangle] less\text{-}than\text{-}char\text{-}linear$
apply $(auto\ simp: Relation.total\text{-}on\text{-}def)$
using $lexord\text{-}irrefl[OF\ irreft\text{-}less\text{-}than\text{-}char]$
 $antisym\text{-}lexord[OF\ antisym\text{-}less\text{-}than\text{-}char\ irreft\text{-}less\text{-}than\text{-}char]$
apply $(auto\ simp: antisym\text{-}def)$
done

lemma $string\text{-}trans:$
 $\langle (xa, ya) \in lexord\ \{(x::char, y::char). x < y\} \implies$
 $(ya, z) \in lexord\ \{(x::char, y::char). x < y\} \implies$
 $(xa, z) \in lexord\ \{(x::char, y::char). x < y\} \rangle$
by $(smt\ less\text{-}char\text{-}def\ char.less\text{-}trans\ less\text{-}than\text{-}char\text{-}def\ lexord\text{-}partial\text{-}trans\ p2rel\text{-}def)$

lemma $full\text{-}quicksort\text{-}sort\text{-}vars\text{-}spec:$
 $\langle (full\text{-}quicksort\text{-}vars, sort\text{-}coeff) \in \langle Id \rangle list\text{-}rel \rightarrow_f \langle \langle Id \rangle list\text{-}rel \rangle nres\text{-}rel \rangle$

proof –
have $xs: \langle (xs, xs) \in \langle Id \rangle list\text{-}rel \rangle$ **and** $\langle \downarrow (\langle Id \rangle list\text{-}rel) x = x \rangle$ **for** $x\ xs$

```

  by auto
show ?thesis
apply (intro frefI nres-relI)
unfolding full-quicksort-vars-def
apply (rule full-quicksort-ref-full-quicksort[THEN fref-to-Down-curry, THEN order-trans])
subgoal
  by (auto simp: rel2p-def var-order-rel-def p2rel-def Relation.total-on-def
      dest: string-trans)
subgoal
  using total-on-lexord-less-than-char-linear2[unfolded var-order-rel-def]
  apply (auto simp: rel2p-def var-order-rel-def p2rel-def Relation.total-on-def less-char-def)
  done
subgoal by fast
apply (rule xs)
apply (subst down-eq-id-list-rel)
unfolding sorted-wrt-map sort-coeff-def
apply (rule full-quicksort-correct-sorted[where  $R = \langle (\lambda x y. x = y \vee (x, y) \in \text{var-order-rel}) \rangle$  and  $h = \langle \text{id} \rangle$ ,
      THEN order-trans])
subgoal
  by (auto simp: rel2p-def var-order-rel-def p2rel-def Relation.total-on-def dest: string-trans)
subgoal for x y
  using total-on-lexord-less-than-char-linear2[unfolded var-order-rel-def]
  by (auto simp: rel2p-def var-order-rel-def p2rel-def Relation.total-on-def
      less-char-def)
subgoal
  by (auto simp: rel2p-def p2rel-def rel2p-def[abs-def])
done
qed

```

```

sepref-definition partition-main-vars-impl
is  $\langle \text{uncurry2 } \text{partition-main-vars} \rangle$ 
::  $\langle \text{nat-assn}^k *_{\alpha} \text{nat-assn}^k *_{\alpha} (\text{monom-assn})^k \rightarrow_{\alpha} \text{prod-assn } (\text{monom-assn}) \text{ nat-assn} \rangle$ 
unfolding partition-main-vars-def partition-main-def
  var-order-rel-var-order
  var-order'-def[symmetric]
  term-order-rel'-alt-def
  le-term-order-rel'
  id-apply
  by sepref

```

```

declare partition-main-vars-impl.refine[sepref-fr-rules]

```

```

sepref-definition partition-between-vars-impl
is  $\langle \text{uncurry2 } \text{partition-between-vars} \rangle$ 
::  $\langle \text{nat-assn}^k *_{\alpha} \text{nat-assn}^k *_{\alpha} \text{monom-assn}^k \rightarrow_{\alpha} \text{prod-assn } \text{monom-assn } \text{nat-assn} \rangle$ 
unfolding partition-between-vars-def partition-between-ref-def
  partition-main-vars-def[symmetric]
unfolding choose-pivot3-def
  term-order-rel'-def[symmetric]
  term-order-rel'-alt-def choose-pivot-def
  le-term-order-rel' id-apply
  by sepref

```

declare *partition-between-vars-impl.refine*[*sepref-fr-rules*]

sepref-definition *quicksort-vars-impl*

is $\langle \text{uncurry2 } \text{quicksort-vars} \rangle$

:: $\langle \text{nat-assn}^k *_{\alpha} \text{nat-assn}^k *_{\alpha} \text{monom-assn}^k \rightarrow_{\alpha} \text{monom-assn} \rangle$

unfolding *partition-main-vars-def quicksort-ref-def quicksort-vars-def*
partition-between-vars-def[*symmetric*]

by *sepref*

lemmas [*sepref-fr-rules*] = *quicksort-vars-impl.refine*

sepref-register *quicksort-vars*

lemma *le-var-order-rel*:

$\langle (\leq) = (\lambda x y. x = y \vee (x, y) \in \text{var-order-rel}) \rangle$

by (*intro ext*)

(*auto simp add: less-list-def less-eq-list-def rel2p-def*
p2rel-def lexordp-conv-lexord p2rel-def var-order-rel-def
lexordp-eq-conv-lexord lexordp-def)

sepref-definition *full-quicksort-vars-impl*

is $\langle \text{full-quicksort-vars} \rangle$

:: $\langle \text{monom-assn}^k \rightarrow_{\alpha} \text{monom-assn} \rangle$

unfolding *full-quicksort-vars-def full-quicksort-ref-def*
quicksort-vars-def[*symmetric*]

le-var-order-rel[*symmetric*]

term-order-rel'-def[*symmetric*]

List.null-def

by *sepref*

lemmas *sort-vars-spec-hnr* =

full-quicksort-vars-impl.refine[*FCOMP full-quicksort-sort-vars-spec*]

lemma *string-rel-order-map*:

$\langle (x, a) \in \text{string-rel} \implies$

$(y, aa) \in \text{string-rel} \implies$

$x \leq y \iff a \leq aa \rangle$

unfolding *string-rel-def less-eq-literal.rep-eq less-than-char-def*
less-eq-list-def PAC-Polynomials-Term.less-char-def[*symmetric*]

by (*auto simp: string-rel-def less-eq-literal.rep-eq less-than-char-def*

less-eq-list-def char.lexordp-eq-conv-lexord lexordp-eq-refl

lexordp-char-char lexordp-eq-conv-lexord

simp flip: less-char-def[*abs-def*])

lemma *merge-monomers-merge-monomers*:

$\langle (\text{merge-monomers}, \text{merge-monomers}) \in \text{monom-rel} \rightarrow \text{monom-rel} \rightarrow \text{monom-rel} \rangle$

unfolding *merge-monomers-def*

apply (*intro fun-relI*)

subgoal for *a a' aa a'a*

apply (*induction* $\langle (\lambda(a :: \text{String.literal})$

$(b :: \text{String.literal}). a \leq b) \rangle a aa$

arbitrary: a' a'a

rule: merge.induct)

```

subgoal
  by (auto elim!: list-relE3 list-relE4 list-relE list-relE2
      simp: string-rel-order-map)
subgoal
  by (auto elim!: list-relE3 list-relE)
subgoal
  by (auto elim!: list-relE3 list-relE4 list-relE list-relE2)
done
done

```

```

lemma merge-monoms-merge-monoms2:
  ⟨(a, b) ∈ monom-rel ⟹ (a', b') ∈ monom-rel ⟹
    (merge-monoms a a', merge-monoms b b') ∈ monom-rel⟩
using merge-monoms-merge-monoms
unfolding fun-rel-def merge-monoms-def
by auto

```

```

lemma msort-monoms-impl:
  ⟨(msort-monoms-impl, merge-monoms-poly)
    ∈ monom-rel → monom-rel⟩
unfolding msort-monoms-impl-def merge-monoms-poly-def
apply (intro fun-relI)
subgoal for a a'
  apply (induction ⟨(λ(a :: String.literal)
    (b :: String.literal). a ≤ b)⟩ a
    arbitrary: a'
    rule: msort.induct)
subgoal
  by auto
subgoal
  by (auto elim!: list-relE3 list-relE)
subgoal premises p
  using p
  by (auto elim!: list-relE3 list-relE4 list-relE list-relE2
      simp: merge-monoms-def[symmetric] intro!: list-rel-takeD list-rel-dropD
      intro!: merge-monoms-merge-monoms2 p(1)[simplified] p(2)[simplified])
      (simp-all add: list-rel-imp-same-length)
done
done

```

```

lemma merge-sort-monoms-sort-monoms-spec:
  ⟨(RETURN o merge-monoms-poly, sort-coeff) ∈ ⟨Id⟩list-rel →f ⟨⟨Id⟩list-rel⟩nres-rel⟩
unfolding merge-monoms-poly-def sort-coeff-def
by (intro frefI nres-relI)
  (auto intro!: sorted-msort simp: sorted-wrt-map rel2p-def
    le-term-order-rel' transp-def rel2p-def[abs-def]
    simp flip: le-var-order-rel)

```

sepref-register sort-coeff

```

lemma [sepref-fr-rules]:
  ⟨(return o msort-monoms-impl, sort-coeff) ∈ monom-assnk →a monom-assn⟩
using msort-monoms-impl[sepref-param, FCOMP merge-sort-monoms-sort-monoms-spec]
by auto

```

sepref-definition *sort-all-coeffs-impl*

```
is ⟨sort-all-coeffs⟩
:: ⟨poly-assnk →a poly-assn⟩
unfolding sort-all-coeffs-def
  HOL-list.fold-custom-empty
by sepref
```

declare *sort-all-coeffs-impl.refine*[sepref-fr-rules]

lemma *merge-coeffs0-alt-def*:

```
⟨(RETURN o merge-coeffs0) p =
  RECT(λf p.
    (case p of
      [] ⇒ RETURN []
    | [p] => if snd p = 0 then RETURN [] else RETURN [p]
    | ((xs, n) # (ys, m) # p) ⇒
      (if xs = ys
        then if n + m ≠ 0 then f ((xs, n + m) # p) else f p
        else if n = 0 then
          do {p ← f ((ys, m) # p);
            RETURN p}
        else do {p ← f ((ys, m) # p);
            RETURN ((xs, n) # p)})))
  p⟩
apply (subst eq-commute)
apply (induction p rule: merge-coeffs0.induct)
subgoal by (subst RECT-unfold, refine-mono) auto
subgoal by (subst RECT-unfold, refine-mono) auto
subgoal by (subst RECT-unfold, refine-mono) (auto simp: let-to-bind-conv)
done
```

Again, Sepref does not understand what is going here.

sepref-definition *merge-coeffs0-impl*

```
is ⟨RETURN o merge-coeffs0⟩
:: ⟨poly-assnk →a poly-assn⟩
supply [[goals-limit=1]]
unfolding merge-coeffs0-alt-def
  HOL-list.fold-custom-empty
apply sepref-dbg-preproc
apply sepref-dbg-cons-init
apply sepref-dbg-id
apply sepref-dbg-monadify
apply sepref-dbg-opt-init
apply (rule WTF-RF | sepref-dbg-trans-step)+
apply sepref-dbg-opt
apply sepref-dbg-cons-solve
apply sepref-dbg-cons-solve
apply sepref-dbg-constraints
done
```

declare *merge-coeffs0-impl.refine*[sepref-fr-rules]

sepref-definition *fully-normalize-poly-impl*

```
is ⟨full-normalize-poly⟩
```

```

:: ⟨poly-assnk →a poly-assn⟩
supply [[goals-limit=1]]
unfolding full-normalize-poly-def
by sepref

declare fully-normalize-poly-impl.refine[sepref-fr-rules]

end

```

```

theory PAC-Version
  imports Main
begin

```

This code was taken from IsaFoR. However, for the AFP, we use the version name *AFP*, instead of a mercurial version.

```

local-setup ⟨
  let
    val version = AFP
  in
    Local-Theory.define
      ((binding ⟨version⟩, NoSyn),
       ((binding ⟨version-def⟩, []), HOLogic.mk-literal version)) #> #2
  end
⟩

```

```

declare version-def [code]

```

```

end

```

```

theory PAC-Checker-Synthesis
  imports PAC-Checker WB-Sort PAC-Checker-Relation
    PAC-Checker-Init More-Loops PAC-Version
begin

```

13 Code Synthesis of the Complete Checker

We here combine refine the full checker, using the initialisation provided in another file and adding more efficient data structures (mostly replacing the set of variables by a more efficient hash map).

```

abbreviation vars-assn where
  ⟨vars-assn ≡ hs.assn string-assn⟩

```

```

fun vars-of-monom-in where
  ⟨vars-of-monom-in [] = True⟩ |
  ⟨vars-of-monom-in (x # xs) V ⟷ x ∈ V ∧ vars-of-monom-in xs V⟩

```

```

fun vars-of-poly-in where
  ⟨vars-of-poly-in [] = True⟩ |
  ⟨vars-of-poly-in ((x, -) # xs) V ⟷ vars-of-monom-in x V ∧ vars-of-poly-in xs V⟩

```

```

lemma vars-of-monom-in-alt-def:
  ⟨vars-of-monom-in xs V ⟷ set xs ⊆ V⟩
by (induction xs)

```

auto

lemma *vars-llist-alt-def*:
⟨*vars-llist* $xs \subseteq \mathcal{V} \longleftrightarrow \text{vars-of-poly-in } xs \ \mathcal{V}$ ⟩
by (*induction xs*)
(*auto simp: vars-llist-def vars-of-monom-in-alt-def*)

lemma *vars-of-monom-in-alt-def2*:
⟨*vars-of-monom-in* $xs \ \mathcal{V} \longleftrightarrow \text{fold } (\lambda x b. b \wedge x \in \mathcal{V}) \ xs \ \text{True}$ ⟩
apply (*subst foldr-fold[symmetric]*)
subgoal by *auto*
subgoal by (*induction xs*) *auto*
done

sempref-definition *vars-of-monom-in-impl*
is ⟨*uncurry* (*RETURN oo vars-of-monom-in*)⟩
:: ⟨(*list-assn string-assn*)^{*k*} *_{*a*} *vars-assn*^{*k*} →_{*a*} *bool-assn*⟩
unfolding *vars-of-monom-in-alt-def2*
by *sempref*

declare *vars-of-monom-in-impl.refine[sempref-fr-rules]*

lemma *vars-of-poly-in-alt-def2*:
⟨*vars-of-poly-in* $xs \ \mathcal{V} \longleftrightarrow \text{fold } (\lambda(x, -) b. b \wedge \text{vars-of-monom-in } x \ \mathcal{V}) \ xs \ \text{True}$ ⟩
apply (*subst foldr-fold[symmetric]*)
subgoal by *auto*
subgoal by (*induction xs*) *auto*
done

sempref-definition *vars-of-poly-in-impl*
is ⟨*uncurry* (*RETURN oo vars-of-poly-in*)⟩
:: ⟨(*poly-assn*)^{*k*} *_{*a*} *vars-assn*^{*k*} →_{*a*} *bool-assn*⟩
unfolding *vars-of-poly-in-alt-def2*
by *sempref*

declare *vars-of-poly-in-impl.refine[sempref-fr-rules]*

definition *union-vars-monom* :: ⟨*string list* ⇒ *string set* ⇒ *string set*⟩ **where**
⟨*union-vars-monom* $xs \ \mathcal{V} = \text{fold insert } xs \ \mathcal{V}$ ⟩

definition *union-vars-poly* :: ⟨*llist-polynomial* ⇒ *string set* ⇒ *string set*⟩ **where**
⟨*union-vars-poly* $xs \ \mathcal{V} = \text{fold } (\lambda(xs, -) \ \mathcal{V}. \text{union-vars-monom } xs \ \mathcal{V}) \ xs \ \mathcal{V}$ ⟩

lemma *union-vars-monom-alt-def*:
⟨*union-vars-monom* $xs \ \mathcal{V} = \mathcal{V} \cup \text{set } xs$ ⟩
unfolding *union-vars-monom-def*
apply (*subst foldr-fold[symmetric]*)
subgoal for $x \ y$
by (*cases x; cases y*) *auto*
subgoal
by (*induction xs*) *auto*
done


```

lemma union-vars-poly-alt-def:
  ⟨union-vars-poly xs  $\mathcal{V} = \mathcal{V} \cup \text{vars-llist } xs$ ⟩
  unfolding union-vars-poly-def
  apply (subst foldr-fold[symmetric])
  subgoal for x y
    by (cases x; cases y)
      (auto simp: union-vars-monom-alt-def)
  subgoal
    by (induction xs)
      (auto simp: vars-llist-def union-vars-monom-alt-def)
  done

```

```

sempref-definition union-vars-monom-impl
  is ⟨uncurry (RETURN oo union-vars-monom)⟩
  :: ⟨monom-assnk *a vars-assnd →a vars-assn⟩
  unfolding union-vars-monom-def
  by sempref

```

```

declare union-vars-monom-impl.refine[sempref-fr-rules]

```

```

sempref-definition union-vars-poly-impl
  is ⟨uncurry (RETURN oo union-vars-poly)⟩
  :: ⟨poly-assnk *a vars-assnd →a vars-assn⟩
  unfolding union-vars-poly-def
  by sempref

```

```

declare union-vars-poly-impl.refine[sempref-fr-rules]

```

```

hide-const (open) Autoref-Fix-Rel.CONSTRAINT

```

```

fun status-assn where
  ⟨status-assn - CSUCCESS CSUCCESS = emp⟩ |
  ⟨status-assn - CFOUND CFOUND = emp⟩ |
  ⟨status-assn R (CFAILED a) (CFAILED b) = R a b⟩ |
  ⟨status-assn - - = false⟩

```

```

lemma SUCCESS-hnr[sempref-fr-rules]:
  ⟨(uncurry0 (return CSUCCESS)), uncurry0 (RETURN CSUCCESS)⟩ ∈ unit-assnk →a status-assn R⟩
  by (sempref-to-hoare)
    sep-auto

```

```

lemma FOUND-hnr[sempref-fr-rules]:
  ⟨(uncurry0 (return CFOUND)), uncurry0 (RETURN CFOUND)⟩ ∈ unit-assnk →a status-assn R⟩
  by (sempref-to-hoare)
    sep-auto

```

```

lemma is-success-hnr[sempref-fr-rules]:
  ⟨CONSTRAINT is-pure R ⇒
  ((return o is-cfound), (RETURN o is-cfound)) ∈ (status-assn R)k →a bool-assn⟩
  apply (sempref-to-hoare)
  apply (rename-tac xi x; case-tac xi; case-tac x)
  apply sep-auto+
  done

```

lemma *is-cfailed-hnr*[*sepref-fr-rules*]:
 ‹*CONSTRAINT is-pure R* \implies
 ((*return o is-cfailed*), (*RETURN o is-cfailed*)) \in (*status-assn R*)^k \rightarrow_a *bool-assn*›
apply (*sepref-to-hoare*)
apply (*rename-tac xi x*; *case-tac xi*; *case-tac x*)
apply *sep-auto+*
done

lemma *merge-cstatus-hnr*[*sepref-fr-rules*]:
 ‹*CONSTRAINT is-pure R* \implies
 (*uncurry (return oo merge-cstatus)*, *uncurry (RETURN oo merge-cstatus)*) \in
 (*status-assn R*)^k *_a (*status-assn R*)^k \rightarrow_a *status-assn R*›
apply (*sepref-to-hoare*)
by (*case-tac b*; *case-tac bi*; *case-tac a*; *case-tac ai*; *sep-auto simp: is-pure-conv pure-app-eq*)

sepref-definition *add-poly-impl*
is ‹*add-poly-l*›
 :: ‹(*poly-assn* \times_a *poly-assn*)^k \rightarrow_a *poly-assn*›
supply [[*goals-limit=1*]]
unfolding *add-poly-l-def*
 HOL-list.fold-custom-empty
 term-order-rel'-def[symmetric]
 term-order-rel'-alt-def
by *sepref*

declare *add-poly-impl.refine*[*sepref-fr-rules*]

sepref-register *mult-monomials*

lemma *mult-monomials-alt-def*:
 ‹(*RETURN oo mult-monomials*) *x y = RECT*
 (λf (*p, q*).
 case (p, q) of
 (\square , -) \Rightarrow *RETURN q*
 | (-, \square) \Rightarrow *RETURN p*
 | (*x # p, y # q*) \Rightarrow
 (*if x = y then do* {
 pq \leftarrow *f (p, q)*;
 RETURN (x # pq)}
 else if (x, y) \in var-order-rel
 then do {
 pq \leftarrow *f (p, y # q)*;
 RETURN (x # pq)}
 else do {
 pq \leftarrow *f (x # p, q)*;
 RETURN (y # pq)}
 }))
 (*x, y*)›
apply (*subst eq-commute*)
apply (*induction x y rule: mult-monomials.induct*)
subgoal for *p*
by (*subst RECT-unfold, refine-mono*) (*auto split: list.splits*)
subgoal for *p*
by (*subst RECT-unfold, refine-mono*) (*auto split: list.splits*)
subgoal for *x p y q*

by (subst RECT-unfold, refine-mono) (auto split: list.splits simp: let-to-bind-conv)
done

sepref-definition *mult-monomials-impl*
is $\langle \text{uncurry } (\text{RETURN } \text{oo } \text{mult-monomials}) \rangle$
 $:: \langle (\text{monom-assn})^k *_{\alpha} (\text{monom-assn})^k \rightarrow_{\alpha} (\text{monom-assn}) \rangle$
supply [[goals-limit=1]]
unfolding *mult-poly-raw-def*
HOL-list.fold-custom-empty
var-order'-def[symmetric]
term-order-rel'-alt-def
mult-monomials-alt-def
var-order-rel-var-order
by *sepref*

declare *mult-monomials-impl.refine[sepref-fr-rules]*

sepref-definition *mult-monomials-impl*
is $\langle \text{uncurry } (\text{RETURN } \text{oo } \text{mult-monomials}) \rangle$
 $:: \langle (\text{monomial-assn})^k *_{\alpha} (\text{monomial-assn})^k \rightarrow_{\alpha} (\text{monomial-assn}) \rangle$
supply [[goals-limit=1]]
unfolding *mult-monomials-def*
HOL-list.fold-custom-empty
term-order-rel'-def[symmetric]
term-order-rel'-alt-def
by *sepref*

lemma *map-append-alt-def2*:
 $\langle (\text{RETURN } \text{o } (\text{map-append } f \ b)) \ x \rangle = \text{RECT}$
 $(\lambda g \ x \ s. \ \text{case } x \ \text{of } [] \Rightarrow \text{RETURN } b$
 $\quad | \ x \ \# \ x \ s \Rightarrow \ \text{do } \{$
 $\quad \quad y \leftarrow g \ x \ s;$
 $\quad \quad \text{RETURN } (f \ x \ \# \ y)$
 $\quad \}) \ x \ s \rangle$
apply (*subst eq-commute*)
apply (*induction f b xs rule: map-append.induct*)
subgoal **by** (*subst RECT-unfold, refine-mono*) *auto*
subgoal **by** (*subst RECT-unfold, refine-mono*) *auto*
done

definition *map-append-poly-mult* **where**
 $\langle \text{map-append-poly-mult } x = \text{map-append } (\text{mult-monomials } x) \rangle$

declare *mult-monomials-impl.refine[sepref-fr-rules]*

sepref-definition *map-append-poly-mult-impl*
is $\langle \text{uncurry2 } (\text{RETURN } \text{ooo } \text{map-append-poly-mult}) \rangle$
 $:: \langle (\text{monomial-assn})^k *_{\alpha} (\text{poly-assn})^k *_{\alpha} (\text{poly-assn})^k \rightarrow_{\alpha} (\text{poly-assn}) \rangle$
unfolding *map-append-poly-mult-def*
map-append-alt-def2
by *sepref*

declare *map-append-poly-mult-impl.refine*[*sepref-fr-rules*]

TODO *foldl* ($\lambda x. l @ [?f x]$) [] ?l = *map* ?f ?l is the worst possible implementation of *map*!

sepref-definition *mult-poly-raw-impl*
is $\langle \text{uncurry } (RETURN \text{ oo } \text{mult-poly-raw}) \rangle$
:: $\langle \text{poly-assn}^k *_a \text{poly-assn}^k \rightarrow_a \text{poly-assn} \rangle$
supply [[*goals-limit=1*]]
supply [[*eta-contract = false, show-abbrevs=false*]]
unfolding *mult-poly-raw-def*
 HOL-list.fold-custom-empty
 term-order-rel'-def[*symmetric*]
 term-order-rel'-alt-def
 foldl-conv-fold
 fold-eq-nfoldli
 map-append-poly-mult-def[*symmetric*]
 map-append-alt-def[*symmetric*]
by *sepref*

declare *mult-poly-raw-impl.refine*[*sepref-fr-rules*]

sepref-definition *mult-poly-impl*
is $\langle \text{uncurry } \text{mult-poly-full} \rangle$
:: $\langle \text{poly-assn}^k *_a \text{poly-assn}^k \rightarrow_a \text{poly-assn} \rangle$
supply [[*goals-limit=1*]]
unfolding *mult-poly-full-def*
 HOL-list.fold-custom-empty
 term-order-rel'-def[*symmetric*]
 term-order-rel'-alt-def
by *sepref*

declare *mult-poly-impl.refine*[*sepref-fr-rules*]

lemma *inverse-monomial*:
 $\langle \text{monom-rel}^{-1} \times_r \text{int-rel} = (\text{monom-rel} \times_r \text{int-rel})^{-1} \rangle$
by (*auto*)

lemma *eq-poly-rel-eq*[*sepref-import-param*]:
 $\langle ((=), (=)) \in \text{poly-rel} \rightarrow \text{poly-rel} \rightarrow \text{bool-rel} \rangle$
using *list-rel-sv*[*of* $\langle \text{monomial-rel} \rangle$, *OF single-valued-monomial-rel*]
using *list-rel-sv*[*OF single-valued-monomial-rel'*[*unfolded IS-LEFT-UNIQUE-def inv-list-rel-eq*]]
unfolding *inv-list-rel-eq*[*symmetric*]
by (*auto intro!*: *freqI simp*:
 rel2p-def single-valued-def p2rel-def
 simp del: inv-list-rel-eq)

sepref-definition *weak-equality-l-impl*
is $\langle \text{uncurry } \text{weak-equality-l} \rangle$
:: $\langle \text{poly-assn}^k *_a \text{poly-assn}^k \rightarrow_a \text{bool-assn} \rangle$
supply [[*goals-limit=1*]]
unfolding *weak-equality-l-def*
by *sepref*

declare *weak-equality-l-impl.refine*[*sepref-fr-rules*]
sepref-register *add-poly-l mult-poly-full*

abbreviation *raw-string-assn* :: $\langle \text{string} \Rightarrow \text{string} \Rightarrow \text{assn} \rangle$ **where**
 $\langle \text{raw-string-assn} \equiv \text{list-assn id-assn} \rangle$

definition *show-nat* :: $\langle \text{nat} \Rightarrow \text{string} \rangle$ **where**
 $\langle \text{show-nat } i = \text{show } i \rangle$

lemma [*sepref-import-param*]:
 $\langle (\text{show-nat}, \text{show-nat}) \in \text{nat-rel} \rightarrow \langle \text{Id} \rangle \text{list-rel} \rangle$
by (*auto intro: fun-relI*)

lemma *status-assn-pure-conv*:
 $\langle \text{status-assn } (\text{id-assn}) \ a \ b = \text{id-assn } a \ b \rangle$
by (*cases a; cases b*)
(auto simp: pure-def)

lemma [*sepref-fr-rules*]:
 $\langle (\text{uncurry3 } (\lambda x y. \text{return } oo (\text{error-msg-not-equal-dom } x \ y)), \text{uncurry3 } \text{check-not-equal-dom-err}) \in$
 $\text{poly-assn}^k *_{\mathbf{a}} \text{poly-assn}^k *_{\mathbf{a}} \text{poly-assn}^k *_{\mathbf{a}} \text{poly-assn}^k \rightarrow_{\mathbf{a}} \text{raw-string-assn} \rangle$
unfolding *show-nat-def[symmetric]* *list-assn-pure-conv*
prod-assn-pure-conv *check-not-equal-dom-err-def*
by (*sepref-to-hoare; sep-auto simp: error-msg-not-equal-dom-def*)

lemma [*sepref-fr-rules*]:
 $\langle (\text{return } o (\text{error-msg-notin-dom } o \ \text{nat-of-uint64}), \text{RETURN } o \ \text{error-msg-notin-dom})$
 $\in \text{uint64-nat-assn}^k \rightarrow_{\mathbf{a}} \text{raw-string-assn} \rangle$
 $\langle (\text{return } o (\text{error-msg-reused-dom } o \ \text{nat-of-uint64}), \text{RETURN } o \ \text{error-msg-reused-dom})$
 $\in \text{uint64-nat-assn}^k \rightarrow_{\mathbf{a}} \text{raw-string-assn} \rangle$
 $\langle (\text{uncurry } (\text{return } oo (\lambda i. \text{error-msg } (\text{nat-of-uint64 } \ i))), \text{uncurry } (\text{RETURN } oo \ \text{error-msg}))$
 $\in \text{uint64-nat-assn}^k *_{\mathbf{a}} \text{raw-string-assn}^k \rightarrow_{\mathbf{a}} \text{status-assn } \text{raw-string-assn} \rangle$
 $\langle (\text{uncurry } (\text{return } oo \ \text{error-msg}), \text{uncurry } (\text{RETURN } oo \ \text{error-msg}))$
 $\in \text{nat-assn}^k *_{\mathbf{a}} \text{raw-string-assn}^k \rightarrow_{\mathbf{a}} \text{status-assn } \text{raw-string-assn} \rangle$
unfolding *error-msg-notin-dom-def* *list-assn-pure-conv* *list-rel-id-simp*
unfolding *status-assn-pure-conv*
unfolding *show-nat-def[symmetric]*
by (*sepref-to-hoare; sep-auto simp: uint64-nat-rel-def br-def; fail*)+

sepref-definition *check-addition-l-impl*
is $\langle \text{uncurry6 } \text{check-addition-l} \rangle$
 $\langle \text{poly-assn}^k *_{\mathbf{a}} \text{polys-assn}^k *_{\mathbf{a}} \text{vars-assn}^k *_{\mathbf{a}} \text{uint64-nat-assn}^k *_{\mathbf{a}} \text{uint64-nat-assn}^k *_{\mathbf{a}}$
 $\text{uint64-nat-assn}^k *_{\mathbf{a}} \text{poly-assn}^k \rightarrow_{\mathbf{a}} \text{status-assn } \text{raw-string-assn} \rangle$
supply $[[\text{goals-limit}=1]]$
unfolding *mult-poly-full-def*
HOL-list.fold-custom-empty
term-order-rel'-def[symmetric]
term-order-rel'-alt-def
check-addition-l-def
in-dom-m-lookup-iff
fmlookup'-def[symmetric]
vars-llist-alt-def
by *sepref*

declare *check-addition-l-impl.refine*[*sepref-fr-rules*]

sepref-register *check-mult-l-dom-err*

definition *check-mult-l-dom-err-impl* **where**

⟨*check-mult-l-dom-err-impl* *pd p ia i* =
 (if *pd* then "The polynomial with id " @ show (nat-of-uint64 *p*) @ " was not found" else "") @
 (if *ia* then "The id of the resulting id " @ show (nat-of-uint64 *i*) @ " was already given" else "")⟩

definition *check-mult-l-mult-err-impl* **where**

⟨*check-mult-l-mult-err-impl* *p q pq r* =
 "Multiplying " @ show *p* @ " by " @ show *q* @ " gives " @ show *pq* @ " and not " @ show *r*⟩

lemma [*sepref-fr-rules*]:

⟨(uncurry3 ((λ*x y*. return oo (*check-mult-l-dom-err-impl* *x y*))),
 uncurry3 (*check-mult-l-dom-err*)) ∈ bool-assn^k *_a uint64-nat-assn^k *_a bool-assn^k *_a uint64-nat-assn^k
→_a raw-string-assn⟩
unfolding *check-mult-l-dom-err-def check-mult-l-dom-err-impl-def list-assn-pure-conv*
apply *sepref-to-hoare*
apply *sep-auto*
done

lemma [*sepref-fr-rules*]:

⟨(uncurry3 ((λ*x y*. return oo (*check-mult-l-mult-err-impl* *x y*))),
 uncurry3 (*check-mult-l-mult-err*)) ∈ poly-assn^k *_a poly-assn^k *_a poly-assn^k *_a poly-assn^k →_a raw-string-assn⟩
unfolding *check-mult-l-mult-err-def check-mult-l-mult-err-impl-def list-assn-pure-conv*
apply *sepref-to-hoare*
apply *sep-auto*
done

sepref-definition *check-mult-l-impl*

is ⟨uncurry6 *check-mult-l*⟩
:: ⟨poly-assn^k *_a polys-assn^k *_a vars-assn^k *_a uint64-nat-assn^k *_a poly-assn^k *_a uint64-nat-assn^k *_a
poly-assn^k →_a status-assn raw-string-assn⟩
supply [[*goals-limit=1*]]
unfolding *check-mult-l-def*
HOL-list.fold-custom-empty
term-order-rel'-def[symmetric]
term-order-rel'-alt-def
in-dom-m-lookup-iff
fmlookup'-def[symmetric]
vars-llist-alt-def
by *sepref*

declare *check-mult-l-impl.refine*[*sepref-fr-rules*]

definition *check-ext-l-dom-err-impl* :: ⟨uint64 ⇒ -⟩ **where**

⟨*check-ext-l-dom-err-impl* *p* =
 "There is already a polynomial with index " @ show (nat-of-uint64 *p*)⟩

lemma [*sepref-fr-rules*]:

⟨(((return o (*check-ext-l-dom-err-impl*))),
 (*check-extension-l-dom-err*)) ∈ uint64-nat-assn^k →_a raw-string-assn⟩
unfolding *check-extension-l-dom-err-def check-ext-l-dom-err-impl-def list-assn-pure-conv*
apply *sepref-to-hoare*

apply *sep-auto*
done

definition *check-extension-l-no-new-var-err-impl* :: $\langle - \Rightarrow - \rangle$ **where**

\langle *check-extension-l-no-new-var-err-impl* *p* =
 "*No new variable could be found in polynomial* " @ *show p* \rangle

lemma [*sepref-fr-rules*]:

\langle ((*return* *o* (*check-extension-l-no-new-var-err-impl*))),
 (*check-extension-l-no-new-var-err*) \in *poly-assn*^{*k*} \rightarrow_a *raw-string-assn* \rangle
unfolding *check-extension-l-no-new-var-err-impl-def* *check-extension-l-no-new-var-err-def*
 list-assn-pure-conv
apply *sepref-to-hoare*
apply *sep-auto*
done

definition *check-extension-l-side-cond-err-impl* :: $\langle - \Rightarrow - \rangle$ **where**

\langle *check-extension-l-side-cond-err-impl* *v p r s* =
 "*Error while checking side conditions of extensions polynow, var is* " @ *show v* @
 "*polynomial is* " @ *show p* @ "*side condition p*p - p =* " @ *show s* @ "*and should be 0*" \rangle

lemma [*sepref-fr-rules*]:

\langle ((*uncurry3* ($\lambda x y.$ *return oo* (*check-extension-l-side-cond-err-impl* *x y*))),
 uncurry3 (*check-extension-l-side-cond-err*) \in *string-assn*^{*k*} *_{*a*} *poly-assn*^{*k*} *_{*a*} *poly-assn*^{*k*} *_{*a*} *poly-assn*^{*k*}
 \rightarrow_a *raw-string-assn* \rangle
unfolding *check-extension-l-side-cond-err-impl-def* *check-extension-l-side-cond-err-def*
 list-assn-pure-conv
apply *sepref-to-hoare*
apply *sep-auto*
done

definition *check-extension-l-new-var-multiple-err-impl* :: $\langle - \Rightarrow - \rangle$ **where**

\langle *check-extension-l-new-var-multiple-err-impl* *v p* =
 "*Error while checking side conditions of extensions polynow, var is* " @ *show v* @
 "*but it either appears at least once in the polynomial or another new variable is created* " @
 show p @ "*but should not.*" \rangle

lemma [*sepref-fr-rules*]:

\langle ((*uncurry* (*return oo* (*check-extension-l-new-var-multiple-err-impl*))),
 uncurry (*check-extension-l-new-var-multiple-err*) \in *string-assn*^{*k*} *_{*a*} *poly-assn*^{*k*} \rightarrow_a *raw-string-assn* \rangle
unfolding *check-extension-l-new-var-multiple-err-impl-def*
 check-extension-l-new-var-multiple-err-def
 list-assn-pure-conv
apply *sepref-to-hoare*
apply *sep-auto*
done

sepref-register *check-extension-l-dom-err* *fmlookup'*
 check-extension-l-side-cond-err *check-extension-l-no-new-var-err*
 check-extension-l-new-var-multiple-err

definition *uminus-poly* :: \langle *l**list-polynomial* \Rightarrow *l**list-polynomial* \rangle **where**

\langle *uminus-poly* *p'* = *map* ($\lambda(a, b).$ (*a*, - *b*)) *p'* \rangle

sepref-register *uminus-poly*

lemma [*sepref-import-param*]:

$\langle \text{map } (\lambda(a, b). (a, - b)), \text{uminus-poly} \rangle \in \text{poly-rel} \rightarrow \text{poly-rel}$

unfolding *uminus-poly-def*

apply (*intro fun-relI*)

subgoal for *p p'*

by (*induction p p' rule: list-rel-induct*)

auto

done

sepref-register *vars-of-poly-in*

weak-equality-l

lemma [*safe-constraint-rules*]:

$\langle \text{Sepref-Constraints.CONSTRAINT single-valued (the-pure monomial-assn)} \rangle$ **and**

single-valued-the-monomial-assn:

$\langle \text{single-valued (the-pure monomial-assn)} \rangle$

$\langle \text{single-valued ((the-pure monomial-assn)}^{-1}) \rangle$

unfolding *IS-LEFT-UNIQUE-def[symmetric]*

by (*auto simp: step-rewrite-pure single-valued-monomial-rel single-valued-monomial-rel' Sepref-Constraints.CONSTRAINT*)

sepref-definition *check-extension-l-impl*

is $\langle \text{uncurry5 check-extension-l} \rangle$

$:: \langle \text{poly-assn}^k *_a \text{polys-assn}^k *_a \text{vars-assn}^k *_a \text{uint64-nat-assn}^k *_a \text{string-assn}^k *_a \text{poly-assn}^k \rightarrow_a \text{status-assn raw-string-assn} \rangle$

supply *option.splits[split] single-valued-the-monomial-assn[simp]*

supply $[[\text{goals-limit}=1]]$

unfolding

HOL-list.fold-custom-empty

term-order-rel'-def[symmetric]

term-order-rel'-alt-def

in-dom-m-lookup-iff

fmlookup'-def[symmetric]

vars-llist-alt-def

check-extension-l-def

not-not

option.case-eq-if

uminus-poly-def[symmetric]

HOL-list.fold-custom-empty

by *sepref*

declare *check-extension-l-impl.refine[sepref-fr-rules]*

sepref-definition *check-del-l-impl*

is $\langle \text{uncurry2 check-del-l} \rangle$

$:: \langle \text{poly-assn}^k *_a \text{polys-assn}^k *_a \text{uint64-nat-assn}^k \rightarrow_a \text{status-assn raw-string-assn} \rangle$

supply $[[\text{goals-limit}=1]]$

unfolding *check-del-l-def*

in-dom-m-lookup-iff

fmlookup'-def[symmetric]

by *sepref*

lemmas [*sepref-fr-rules*] = *check-del-l-impl.refine*

abbreviation *pac-step-rel* **where**

$\langle \text{pac-step-rel} \equiv \text{p2rel} (\langle \text{Id}, \langle \text{monomial-rel} \rangle \text{list-rel}, \text{Id} \rangle \text{pac-step-rel-raw}) \rangle$

sepref-register *PAC-Polynomials-Operations.normalize-poly*

*pac-src1 pac-src2 new-id pac-mult case-pac-step check-mult-l
check-addition-l check-del-l check-extension-l*

lemma *pac-step-rel-assn-alt-def2*:

$\langle \text{hn-ctxt} (\text{pac-step-rel-assn} \text{ nat-assn} \text{ poly-assn} \text{ id-assn}) \text{ b bi} =$
 hn-val
 $(\text{p2rel}$
 $((\text{nat-rel}, \text{poly-rel}, \text{Id} :: (\text{string} \times -) \text{set}) \text{pac-step-rel-raw})) \text{ b bi} \rangle$

unfolding *poly-assn-list hn-ctxt-def*

by (*induction nat-assn poly-assn <id-assn :: string \Rightarrow -> b bi rule: pac-step-rel-assn.induct*)
*(auto simp: p2rel-def hn-val-unfold pac-step-rel-raw.simps relAPP-def
pure-app-eq)*

lemma *is-AddD-import[sepref-fr-rules]*:

assumes $\langle \text{CONSTRAINT is-pure } K \rangle$ $\langle \text{CONSTRAINT is-pure } V \rangle$

shows

$\langle (\text{return } o \text{ pac-res}, \text{RETURN } o \text{ pac-res}) \in [\lambda x. \text{is-Add } x \vee \text{is-Mult } x \vee \text{is-Extension } x]_a$
 $(\text{pac-step-rel-assn } K \text{ V } R)^k \rightarrow V \rangle$

$\langle (\text{return } o \text{ pac-src1}, \text{RETURN } o \text{ pac-src1}) \in [\lambda x. \text{is-Add } x \vee \text{is-Mult } x \vee \text{is-Del } x]_a (\text{pac-step-rel-assn}$
 $K \text{ V } R)^k \rightarrow K \rangle$

$\langle (\text{return } o \text{ new-id}, \text{RETURN } o \text{ new-id}) \in [\lambda x. \text{is-Add } x \vee \text{is-Mult } x \vee \text{is-Extension } x]_a (\text{pac-step-rel-assn}$
 $K \text{ V } R)^k \rightarrow K \rangle$

$\langle (\text{return } o \text{ is-Add}, \text{RETURN } o \text{ is-Add}) \in (\text{pac-step-rel-assn } K \text{ V } R)^k \rightarrow_a \text{bool-assn} \rangle$

$\langle (\text{return } o \text{ is-Mult}, \text{RETURN } o \text{ is-Mult}) \in (\text{pac-step-rel-assn } K \text{ V } R)^k \rightarrow_a \text{bool-assn} \rangle$

$\langle (\text{return } o \text{ is-Del}, \text{RETURN } o \text{ is-Del}) \in (\text{pac-step-rel-assn } K \text{ V } R)^k \rightarrow_a \text{bool-assn} \rangle$

$\langle (\text{return } o \text{ is-Extension}, \text{RETURN } o \text{ is-Extension}) \in (\text{pac-step-rel-assn } K \text{ V } R)^k \rightarrow_a \text{bool-assn} \rangle$

using *assms*

by (*sepref-to-hoare; sep-auto simp: pac-step-rel-assn-alt-def is-pure-conv ent-true-drop pure-app-eq
split: pac-step.splits; fail*) $+$

lemma [*sepref-fr-rules*]:

$\langle \text{CONSTRAINT is-pure } K \implies$

$(\text{return } o \text{ pac-src2}, \text{RETURN } o \text{ pac-src2}) \in [\lambda x. \text{is-Add } x]_a (\text{pac-step-rel-assn } K \text{ V } R)^k \rightarrow K \rangle$

$\langle \text{CONSTRAINT is-pure } V \implies$

$(\text{return } o \text{ pac-mult}, \text{RETURN } o \text{ pac-mult}) \in [\lambda x. \text{is-Mult } x]_a (\text{pac-step-rel-assn } K \text{ V } R)^k \rightarrow V \rangle$

$\langle \text{CONSTRAINT is-pure } R \implies$

$(\text{return } o \text{ new-var}, \text{RETURN } o \text{ new-var}) \in [\lambda x. \text{is-Extension } x]_a (\text{pac-step-rel-assn } K \text{ V } R)^k \rightarrow R \rangle$

by (*sepref-to-hoare; sep-auto simp: pac-step-rel-assn-alt-def is-pure-conv ent-true-drop pure-app-eq
split: pac-step.splits; fail*) $+$

lemma *is-Mult-lastI*:

$\langle \neg \text{is-Add } b \implies \neg \text{is-Mult } b \implies \neg \text{is-Extension } b \implies \text{is-Del } b \rangle$

by (*cases b*) *auto*

sepref-register *is-cfailed is-Del*

definition *PAC-checker-l-step'* :: *- where*

$\langle \text{PAC-checker-l-step}' a \text{ b } c \text{ d} = \text{PAC-checker-l-step } a \text{ (b, c, d)} \rangle$

lemma *PAC-checker-l-step-alt-def*:
 ⟨*PAC-checker-l-step* *a bcd e* = (let (*b,c,d*) = *bcd* in *PAC-checker-l-step'* *a b c d e*)⟩
unfolding *PAC-checker-l-step'-def* **by** *auto*

sepref-decl-intf (*'k*) *acode-status* **is** (*'k*) *code-status*
sepref-decl-intf (*'k*, *'b*, *'lbl*) *apac-step* **is** (*'k*, *'b*, *'lbl*) *pac-step*

sepref-register *merge-cstatus full-normalize-poly new-var is-Add*

lemma *poly-rel-the-pure*:
 ⟨*poly-rel* = *the-pure poly-assn*⟩ **and**
nat-rel-the-pure:
 ⟨*nat-rel* = *the-pure nat-assn*⟩ **and**
WTF-RF: ⟨*pure* (*the-pure nat-assn*) = *nat-assn*⟩
unfolding *poly-assn-list*
by *auto*

lemma [*safe-constraint-rules*]:
 ⟨*CONSTRAINT IS-LEFT-UNIQUE uint64-nat-rel*⟩ **and**
single-valued-uint64-nat-rel[*safe-constraint-rules*]:
 ⟨*CONSTRAINT single-valued uint64-nat-rel*⟩
by (*auto simp: IS-LEFT-UNIQUE-def single-valued-def uint64-nat-rel-def br-def*)

sepref-definition *check-step-impl*
is ⟨*uncurry4 PAC-checker-l-step'*⟩
 :: ⟨*poly-assn*^{*k*} *_{*a*} (*status-assn raw-string-assn*)^{*d*} *_{*a*} *vars-assn*^{*d*} *_{*a*} *polys-assn*^{*d*} *_{*a*} (*pac-step-rel-assn*
 (*uint64-nat-assn*) *poly-assn* (*string-assn* :: *string* ⇒ -))^{*d*} →_{*a*}
status-assn raw-string-assn ×_{*a*} *vars-assn* ×_{*a*} *polys-assn*⟩
supply [[*goals-limit=1*]] *is-Mult-lastI*[*intro*] *single-valued-uint64-nat-rel*[*simp*]
unfolding *PAC-checker-l-step-def PAC-checker-l-step'-def*
pac-step.case-eq-if Let-def
is-success-alt-def[*symmetric*]
uminus-poly-def[*symmetric*]
HOL-list.fold-custom-empty
by *sepref*

declare *check-step-impl.refine*[*sepref-fr-rules*]

sepref-register *PAC-checker-l-step PAC-checker-l-step' fully-normalize-poly-impl*

definition *PAC-checker-l'* **where**
 ⟨*PAC-checker-l'* *p* \mathcal{V} *A* *status steps* = *PAC-checker-l* *p* (\mathcal{V} , *A*) *status steps*⟩

lemma *PAC-checker-l-alt-def*:
 ⟨*PAC-checker-l* *p* \mathcal{V} *A* *status steps* =
 (let (\mathcal{V} , *A*) = \mathcal{V} *A* in *PAC-checker-l'* *p* \mathcal{V} *A* *status steps*)⟩
unfolding *PAC-checker-l'-def* **by** *auto*

sepref-definition *PAC-checker-l-impl*
is ⟨*uncurry4 PAC-checker-l'*⟩
 :: ⟨*poly-assn*^{*k*} *_{*a*} *vars-assn*^{*d*} *_{*a*} *polys-assn*^{*d*} *_{*a*} (*status-assn raw-string-assn*)^{*d*} *_{*a*}
 (*list-assn* (*pac-step-rel-assn* (*uint64-nat-assn*) *poly-assn string-assn*))^{*k*} →_{*a*}
status-assn raw-string-assn ×_{*a*} *vars-assn* ×_{*a*} *polys-assn*⟩
supply [[*goals-limit=1*]] *is-Mult-lastI*[*intro*]

unfolding *PAC-checker-l-def is-success-alt-def*[symmetric] *PAC-checker-l-step-alt-def*
nres-bind-let-law[symmetric] *PAC-checker-l'-def*
apply (*subst nres-bind-let-law*)
by *sepref*

declare *PAC-checker-l-impl.refine*[*sepref-fr-rules*]

abbreviation *polys-assn-input* **where**
 $\langle polys-assn-input \equiv iam-fmap-assn \text{ nat-assn } polys-assn \rangle$

definition *remap-polys-l-dom-err-impl* :: $\langle - \rangle$ **where**
 $\langle remap-polys-l-dom-err-impl =$
"Error during initialisation. Too many polynomials where provided. If this happens," @
"please report the example to the authors, because something went wrong during " @
"code generation (code generation to arrays is likely to be broken)." \rangle

lemma [*sepref-fr-rules*]:
 $\langle ((uncurry0 (return (remap-polys-l-dom-err-impl))),$
 $uncurry0 (remap-polys-l-dom-err)) \in unit-assn^k \rightarrow_a raw-string-assn \rangle$
unfolding *remap-polys-l-dom-err-def*
remap-polys-l-dom-err-def
list-assn-pure-conv
by *sepref-to-hoare sep-auto*

MLton is not able to optimise the calls to pow.

lemma *pow-2-64*: $\langle (2::nat) \wedge 64 = 18446744073709551616 \rangle$
by *auto*

sepref-register *upper-bound-on-dom op-fmap-empty*

sepref-definition *remap-polys-l-impl*
is $\langle uncurry2 \text{ remap-polys-l2} \rangle$
 $:: \langle poly-assn^k *_a vars-assn^d *_a polys-assn-input^d \rightarrow_a$
 $status-assn \text{ raw-string-assn } \times_a vars-assn \times_a polys-assn \rangle$
supply [[*goals-limit=1*]] *is-Mult-lastI*[*intro*] *indom-mI*[*dest*]
unfolding *remap-polys-l2-def op-fmap-empty-def*[symmetric] *while-eq-nfoldli*[symmetric]
while-upt-while-direct pow-2-64
in-dom-m-lookup-iff
fmlookup'-def[symmetric]
union-vars-poly-alt-def[symmetric]
apply (*rewrite at* $\langle fmupd \sqsupset \rangle$ *uint64-of-nat-conv-def*[symmetric])
apply (*subst while-upt-while-direct*)
apply *simp*
apply (*rewrite at* $\langle op-fmap-empty \rangle$ *annotate-assn*[**where** $A = \langle polys-assn \rangle$])
by *sepref*

lemma *remap-polys-l2-remap-polys-l*:
 $\langle (uncurry2 \text{ remap-polys-l2}, uncurry2 \text{ remap-polys-l}) \in (Id \times_r \langle Id \rangle set-rel) \times_r Id \rightarrow_f \langle Id \rangle nres-rel \rangle$
apply (*intro frefI fun-reII nres-reII*)
using *remap-polys-l2-remap-polys-l* **by** *auto*

lemma [*sepref-fr-rules*]:
 $\langle (uncurry2 \text{ remap-polys-l-impl},$
 $uncurry2 \text{ remap-polys-l}) \in poly-assn^k *_a vars-assn^d *_a polys-assn-input^d \rightarrow_a$
 $status-assn \text{ raw-string-assn } \times_a vars-assn \times_a polys-assn \rangle$

```

using hfcomp-tcomp-pre[OF remap-polys-l2-remap-polys-l remap-polys-l-impl.refine]
by (auto simp: hrp-comp-def hfprod-def)

```

sempref-register *remap-polys-l*

sempref-definition *full-checker-l-impl*

```

is ⟨uncurry2 full-checker-l⟩
:: ⟨poly-assnk *a polys-assn-inputd *a (list-assn (pac-step-rel-assn (uint64-nat-assn) poly-assn string-assn))k
→a
  status-assn raw-string-assn ×a vars-assn ×a polys-assn⟩
supply [[goals-limit=1]] is-Mult-lastI[intro]
unfolding full-checker-l-def hs.fold-custom-empty
  union-vars-poly-alt-def[symmetric]
  PAC-checker-l-alt-def
by sempref

```

sempref-definition *PAC-update-impl*

```

is ⟨uncurry2 (RETURN ooo fmupd)⟩
:: ⟨nat-assnk *a poly-assnk *a (polys-assn-input)d →a polys-assn-input⟩
unfolding comp-def
by sempref

```

sempref-definition *PAC-empty-impl*

```

is ⟨uncurry0 (RETURN fmempty)⟩
:: ⟨unit-assnk →a polys-assn-input⟩
unfolding op-iam-fmap-empty-def[symmetric] pat-fmap-empty
by sempref

```

sempref-definition *empty-vars-impl*

```

is ⟨uncurry0 (RETURN {})⟩
:: ⟨unit-assnk →a vars-assn⟩
unfolding hs.fold-custom-empty
by sempref

```

This is a hack for performance. There is no need to recheck that that a char is valid when working on chars coming from strings... It is not that important in most cases, but in our case the performance difference is really large.

definition *unsafe-asciis-of-literal* :: ⟨-⟩ **where**

```

⟨unsafe-asciis-of-literal xs = String.asciis-of-literal xs⟩

```

definition *unsafe-asciis-of-literal'* :: ⟨-⟩ **where**

```

[simp, symmetric, code]: ⟨unsafe-asciis-of-literal' = unsafe-asciis-of-literal⟩

```

code-printing

```

constant unsafe-asciis-of-literal' ↪
  (SML) !(List.map (fn c => let val k = Char.ord c in IntInf.fromInt k end) /o String.explode)

```

Now comes the big and ugly and unsafe hack.

Basically, we try to avoid the conversion to IntInf when calculating the hash. The performance gain is roughly 40%, which is a LOT and definitively something we need to do. We are aware that the SML semantic encourages compilers to optimise conversions, but this does not happen here, corroborating our early observation on the verified SAT solver IsaSAT.x

definition *raw-explode* **where**

```

[simp]: ⟨raw-explode = String.explode⟩

```

code-printing

constant *raw-explode* \rightarrow
(SML) *String.explode*

definition \langle *hashcode-literal' s* \equiv
foldl ($\lambda h x. h * 33 + \text{uint32-of-int (of-char } x)$) 5381
(raw-explode s) \rangle

lemmas [*code*] =
hashcode-literal-def[*unfolded String.explode-code*
unsafe-asciis-of-literal-def[*symmetric*]]

definition *uint32-of-char* **where**
[*symmetric, code-unfold*]: \langle *uint32-of-char x = uint32-of-int (int-of-char x)* \rangle

code-printing

constant *uint32-of-char* \rightarrow
(SML) $!(\text{Word32.fromInt } /o (\text{Char.ord}))$

lemma [*code*]: \langle *hashcode s = hashcode-literal' s* \rangle
unfolding *hashcode-literal-def hashcode-list-def*
apply (*auto simp: unsafe-asciis-of-literal-def hashcode-list-def*
String.asciis-of-literal-def hashcode-literal-def hashcode-literal'-def)
done

We compile Pastèque in `PAC_Checker_MLton.thy`.

export-code *PAC-checker-l-impl PAC-update-impl PAC-empty-impl the-error-is-cfailed-is-cfound*
int-of-integer Del Add Mult nat-of-integer String.implode remap-polys-l-impl
fully-normalize-poly-impl union-vars-poly-impl empty-vars-impl
full-checker-l-impl check-step-impl CSUCCESS
Extension hashcode-literal' version
in *SML-imp module-name PAC-Checker*

14 Correctness theorem

context *poly-embed*
begin

definition *full-poly-assn* **where**
 \langle *full-poly-assn = hr-comp poly-assn (fully-unsorted-poly-rel O mset-poly-rel)* \rangle

definition *full-poly-input-assn* **where**
 \langle *full-poly-input-assn = hr-comp*
(hr-comp polys-assn-input
((nat-rel, fully-unsorted-poly-rel O mset-poly-rel)fmap-rel))
polys-rel \rangle

definition *fully-pac-assn* **where**
 \langle *fully-pac-assn = (list-assn*
(hr-comp (pac-step-rel-assn uint64-nat-assn poly-assn string-assn)
(p2rel
(nat-rel,
fully-unsorted-poly-rel O
mset-poly-rel, var-rel)pac-step-rel-raw)))) \rangle

definition *code-status-assn* **where**

$\langle \text{code-status-assn} = \text{hr-comp} (\text{status-assn raw-string-assn})$
 $\text{code-status-status-rel} \rangle$

definition *full-vars-assn* **where**

$\langle \text{full-vars-assn} = \text{hr-comp} (\text{hs.assn string-assn})$
 $(\langle \text{var-rel} \rangle \text{set-rel}) \rangle$

lemma *polys-rel-full-polys-rel*:

$\langle \text{polys-rel-full} = \text{Id} \times_r \text{polys-rel} \rangle$

by (*auto simp: polys-rel-full-def*)

definition *full-polys-assn* :: $\langle \rightarrow \rangle$ **where**

$\langle \text{full-polys-assn} = \text{hr-comp} (\text{hr-comp polys-assn}$
 $(\langle \text{nat-rel},$
 $\text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle \text{fmap-rel}))$
 $\text{polys-rel} \rangle$

Below is the full correctness theorems. It basically states that:

1. assuming that the input polynomials have no duplicate variables

Then:

1. if the checker returns *CFOUND*, the spec is in the ideal and the PAC file is correct
2. if the checker returns *CSUCCESS*, the PAC file is correct (but there is no information on the spec, aka checking failed)
3. if the checker return *CFAILED err*, then checking failed (and *err might* give you an indication of the error, but the correctness theorem does not say anything about that).

The input parameters are:

4. the specification polynomial represented as a list
5. the input polynomials as hash map (as an array of option polynomial)
6. a representation of the PAC proofs.

lemma *PAC-full-correctness*:

$\langle (\text{uncurry2 full-checker-l-impl},$
 $\text{uncurry2} (\lambda \text{spec } A \text{ -. PAC-checker-specification spec } A))$
 $\in (\text{full-poly-assn})^k *_a (\text{full-poly-input-assn})^d *_a (\text{fully-pac-assn})^k \rightarrow_a \text{hr-comp}$
 $(\text{code-status-assn} \times_a \text{full-vars-assn} \times_a \text{hr-comp polys-assn}$
 $(\langle \text{nat-rel}, \text{sorted-poly-rel } O \text{ mset-poly-rel} \rangle \text{fmap-rel}))$
 $\{((st, G), st', G')\}.$
 $st = st' \wedge (st \neq \text{FAILED} \longrightarrow (G, G') \in \text{Id} \times_r \text{polys-rel}) \rangle$

using

full-checker-l-impl.refine[*FCOMP full-checker-l-full-checker'*,
FCOMP full-checker-spec',
unfolded full-poly-assn-def[*symmetric*]
full-poly-input-assn-def[*symmetric*]
fully-pac-assn-def[*symmetric*]
code-status-assn-def[*symmetric*]

```

    full-vars-assn-def[symmetric]
    polys-rel-full-polys-rel
    hr-comp-prod-conv
    full-polys-assn-def[symmetric]]
    hr-comp-Id2
  by auto

```

It would be more efficient to move the parsing to Isabelle, as this would be more memory efficient (and also reduce the TCB). But now comes the fun part: It cannot work. A stream (of a file) is consumed by side effects. Assume that this would work. The code could look like:

Let (read-file file) f

This code is equal to (in the HOL sense of equality): *let - = read-file file in Let (read-file file) f*

However, as an hypothetical *read-file* changes the underlying stream, we would get the next token. Remark that this is already a weird point of ML compilers. Anyway, I see currently two solutions to this problem:

1. The meta-argument: use it only in the Refinement Framework in a setup where copies are disallowed. Basically, this works because we can express the non-duplication constraints on the type level. However, we cannot forbid people from expressing things directly at the HOL level.
2. On the target language side, model the stream as the stream and the position. Reading takes two arguments. First, the position to read. Second, the stream (and the current position) to read. If the position to read does not match the current position, return an error. This would fit the correctness theorem of the code generation (roughly “if it terminates without exception, the answer is the same”), but it is still unsatisfactory.

end

definition $\varphi :: \langle string \Rightarrow nat \rangle$ **where**
 $\langle \varphi = (SOME \varphi. \text{bij } \varphi) \rangle$

lemma *bij- φ* : $\langle \text{bij } \varphi \rangle$
using *someI*[of $\langle \lambda \varphi :: string \Rightarrow nat. \text{bij } \varphi \rangle$]
unfolding $\varphi\text{-def}$ [*symmetric*]
using *poly-embed-EX*
by *auto*

global-interpretation *PAC*: *poly-embed* **where**
 $\varphi = \varphi$
apply *standard*
apply (use *bij- φ* in $\langle \text{auto simp: } \text{bij-def} \rangle$)
done

The full correctness theorem is $(\text{uncurry2 } \text{full-checker-l-impl}, \text{uncurry2 } (\lambda \text{spec } A -. \text{PAC-checker-specification spec } A)) \in \text{PAC.full-poly-assn}^k *_a \text{PAC.full-poly-input-assn}^d *_a \text{PAC.fully-pac-assn}^k \rightarrow_a \text{hr-comp} (\text{PAC.code-status-assn} \times_a \text{PAC.full-vars-assn} \times_a \text{hr-comp polys-assn } (\langle \text{nat-rel}, \text{sorted-poly-rel } O \text{ PAC.mset-poly-rel} \rangle \text{fmap-rel})) \{((st, G), st', G'). st = st' \wedge (st \neq \text{FAILED} \longrightarrow (G, G') \in \text{Id} \times_r \text{polys-rel})\}$.

end

Acknowledgment

This work is supported by Austrian Science Fund (FWF), NFN S11408-N23 (RiSE), and LIT AI Lab funded by the State of Upper Austria.

References

- [1] D. Kaufmann, M. Fleury, and A. Biere. The proof checkers pacheck and pasteque for the practical algebraic calculus. In O. Strichman and A. Ivrii, editors, *Formal Methods in Computer-Aided Design, FMCAD 2020, September 21-24, 2020*. IEEE, 2020.